CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Herramienta de interacción persona-ordenador para la operación de vehículos aéreos no tripulados

Autora: Yolanda de la Hoz Simón
Tutor: Martín Molina González

MADRID, JUNIO DE 2015

INDICE

RESUMEN

En la presente memoria se describe el trabajo de diseño de una herramienta de interacción persona-ordenador (HMI) para la operación y supervisión de vehículos aéreos no tripulados (UAV).

En primer lugar se hace una introducción a los tipos de UAVs y aplicaciones más comunes, describiendo sus características técnicas y los componentes que integra en el sistema.

Mediante la revisión y análisis de los diferentes niveles de autonomía y las diferentes soluciones de presentación existentes en el mercado, se identifican los modos de operación y componentes principales de la interfaz.

A continuación se describe el diseño final del software de la interfaz y el proceso de desarrollo de la misma, para ello se hace un análisis previo del software robótico sobre el que opera el sistema abordo del UAV y se establecen los enlaces de comunicación entre cada uno de los componentes y los requisitos de integración con el sistema.

Finalmente, se muestran las pruebas que se han realizado para validar la construcción de la herramienta.

ABSTRACT

This report outlines the design and construction of a human-machine interface (HMI), designed to facilitate the supervision and operation with unmanned aerial vehicles (UAV).

First, it is described an introduction to UAVs classification and application fields, reviewing the hardware features and software integration components. In order to define the basic components and operation modes in the general design, a brief review of the different presentation solutions and autonomous levels is described.

As a result, it is presented the final software design, the components details and the system integration requirements. Finally, it is also concluded with some of the tests that have been conducted to validate the design and construction of the human-machine interface.

1 INTRODUCION

En los últimos años el mercado de los UAVs (vehículos aéreos no tripulados) se ha expandido significativamente. Debido a su reducido tamaño y gran movilidad resultan elementos clave en tareas de inspección, vigilancia o rescate, especialmente en zonas donde el acceso por tierra resulta complicado o inseguro para el ser humano.

Hasta hace algunas décadas el diseño de estos vehículos hacía necesaria la presencia de uno o varios pilotos en el aparato que pudieran controlarlos. Estos vehículos eran diseñados principalmente para realizar tareas de reconocimiento y de ataque dentro de un ámbito militar, a través de un operador interno o controlados desde estaciones de control situados en puntos estratégicos.

Actualmente, los avances en la robótica inteligente han incrementado considerablemente la autonomía de estos vehículos, ampliando las aplicaciones de estos vehículos a un ámbito civil donde actúan de manera autónoma realizando tareas específicas en diversas áreas. Sin embargo, el diseño del sistema software a bordo de estos vehículos se encuentra aún en una fase de investigación que debe ser validada y contrastada correctamente antes de ser utilizados sobre un escenario real.

Dentro de este contexto, se hace necesario disponer de herramientas de ayuda al operador de estos vehículos más específicas y adaptadas a las necesidades de uso. Siendo necesario el diseño e implementación de una herramienta que permita al operador del vehículo supervisar y controlar su ejecución y que a su vez ofrezca la posibilidad de validar el sistema en vuelo y asegurar que el vehículo es capaz de cumplir los objetivos de manera satisfactoria y sin poner en riesgo la vida de las personas.

Este trabajo forma parte de las actividades de desarrollo informático en el marco de investigación de UAVs con mayor grado de autonomía en el ámbito civil que se realizan conjuntamente entre el grupo de investigación de Sistemas Inteligentes e Ingeniería del Conocimiento de Departamento de Inteligencia Artificia y el grupo de investigación Computer Vision Group, ambos pertenecientes a la Universidad Politécnica de Madrid.

1.1 Objetivos

El presente trabajo fin de grado tiene como objetivo general el diseño e implementación de una herramienta de interacción persona-ordenador para vehículos aéreos no tripulados que facilite el control de manera remota, la comunicación con el UAV y la supervisión del sistema a bordo del UAV a través del envío de comandos, la visualización de la información de vuelo captada por los sensores y la información percibida por el sistema a bordo del UAV.

En concreto, se pretende utilizar esta herramienta para asegurar que el vehículo cumple con los objetivos propuestos en la competición IARC, realizando las tareas de monitorización, guiado y seguimiento en un área con robot terrestres. La herramienta deberá cumplir, por lo tanto, una serie de características específicas que le permitan al operador supervisar el estado actual de la misión concreta a realizar y del sistema en general.

Los objetivos planteados en el presente trabajo se han orientado a cubrir las diferentes fases de diseño e implementación de la herramienta. El desarrollo del trabajo y objetivos propuestos se han dividido en las siguientes fases:

- **Análisis:** Esta tarea tiene como objetivo la realización de un estudio del dominio del problema y análisis de las diferentes soluciones existentes para la operación con vehículos aéreos no tripulados. Dicho estudio comprende tanto la revisión de las diferentes soluciones de presentación existentes para UAVs y de las funcionalidades que estas ofrecen, como un análisis del entorno de operación con el vehículo, a través del estudio de los diferentes niveles de autonomía e interacción del usuario con el sistema para la identificación de los componentes básicos de la aplicación.

- **Diseño:** En esta fase se trata de realizar la definición y diseño de cada uno de los componentes que forman la interfaz y el diseño general de la misma a través de la especificación de requisitos del sistema.

- **Implementación:** Se trata de elegir un lenguaje de programación y herramientas software que faciliten la programación de la herramienta y que permitan una fácil integración y comunicación con el sistema abordo del UAV.

- **Validación:** Esta tarea tiene como objetivo final la realización de las pruebas necesarias que permitan validar el diseño de la herramienta, así como, la integración y comunicación con el sistema software del UAV a través de un entorno simulado del mismo sobre el escenario de vuelo que presenta la competición IARC.

1.2 Organización de la memoria

Esta memoria se divide en 4 capítulos principales. En el primero de ellos se realiza un análisis del dominio del problema, mediante una descripción del entorno del vehículo aéreo no tripulado y un análisis de las diferentes herramientas existentes para la operación con UAV. Este capítulo tiene como finalidad ofrecer un diseño general con las funcionalidades esenciales que debería incluir la herramienta sirviendo como punto de partida para el desarrollo del diseño final.

En el segundo capítulo se describe el diseño de la arquitectura software del que forma parte la herramienta y factores determinantes en el diseño de la misma. Finalmente se proporciona una descripción detallada del diseño de los componentes que integran el sistema y requisitos del mismo.

A continuación, en el tercer capítulo, se realiza una descripción de la implementación de la herramienta e integración y comunicación de los diferentes componentes con el software en el UAV.

Por último, se presentan los resultados obtenidos de las pruebas realizadas para comprobar el correcto funcionamiento del sistema y las herramientas utilizadas en cada uno de los bloques de prueba.

2 VEHÍCULOS AÉREOS NO TRIPULADOS

En este capítulo se hace una introducción a las características técnicas de los vehículos aéreos no tripulados, se estudian los diferentes niveles de autonomía y se hace una clasificación de los modos de vuelo existentes para posteriormente determinar el grado de interacción de la interfaz y dependencia de cada uno de los componentes del sistema a bordo.

Así mismo, se realiza un estudio general de las soluciones de presentación existentes para la operación de vehículos aéreos no tripulados, analizando las ventajas e inconvenientes que estos ofrecen, permitiendo establecer una guía de estilo que servirá como punto de partida para el desarrollo de la herramienta objeto de este trabajo.

2.1 Categorías de vehículos y su operación

Un vehículo aéreo no tripulado (UAV) es una aeronave que vuela sin tripulación [1], donde el control del vehículo se hace de manera remota o mediante sistemas de vuelo autónomo. Aunque la parte más visible es el UAV, es más correcto considerarlos como sistemas aéreos no tripulados (UAS).

En particular, de acuerdo con el estándar de STANAG 4586 [2] (estándar de interoperabilidad), en un UAS se pueden distinguir 3 principales elementos: el segmento de tierra, el segmento aéreo y el enlace de datos.

En el segmento de tierra, es de especial importancia la estación de control de tierra, desde la cual los operadores pueden tomar el control de uno o más vehículos aéreos. Está estación de control suele estar formada por una estación para planificar la misión y otra estación para la exploración o análisis de la información de vuelo. El equipamiento de lanzamiento y recuperación, en cambio, es diseñado específicamente para UAVs de pequeñas dimensiones que no son capaces de realizar un despegue o aterrizaje convencional. Finalmente el equipo de logística asegura la operatividad del UAV como podrían ser por ejemplo las unidades de carga.

El segmento aéreo, en cambio, está formado por el vehículo aéreo no tripulado y su respectivo cargamento.

El subsistema encargado de interconectar ambos segmentos sería el enlace de datos, dividido en el enlace de datos terrestre y aéreo. La mayoría de los vehículos presentan dos enlaces de datos para cada lado de la comunicación; uno para el control y envío de comandos y otro para la recepción de la información de vuelo. Sin embargo, también es

posible encontrar enlaces de comunicación para elementos externos como sería la comunicación satélite mediante GPS o la comunicación con un mando radio control.

Independientemente del tipo de enlace de datos utilizado se distinguen los dos siguientes canales:

- ***Uplink*.** La información es enviada desde el segmento aéreo al segmento terrestre.
- ***Downlink*.** La información es enviada desde el segmento terrestre al segmento aéreo.

A continuación se muestra una figura representativa de los principales elementos en un UAS:

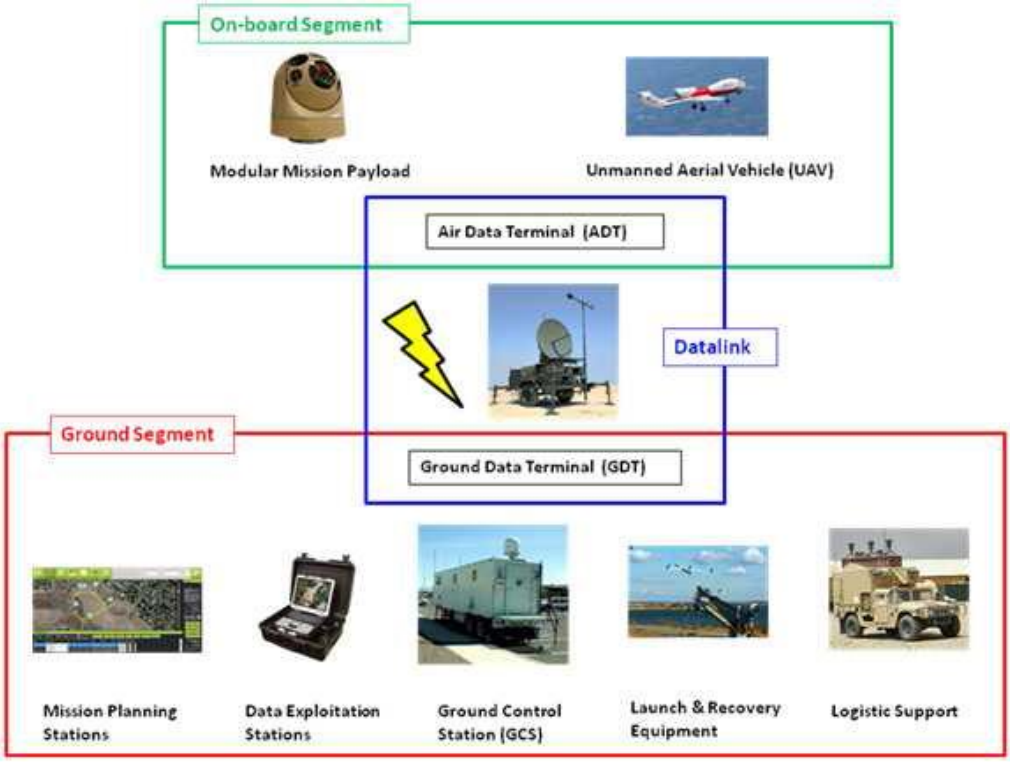


Figura 2-1. Elementos que componen un UAS.

2.1.1 Clasificación de los vehículos aéreos no tripulados

Actualmente no hay un estándar universal aceptado en cuanto a clasificación de UAV, pudiendo encontrar multitud de categorías de vehículos aéreos no tripulados siguiendo diferentes parámetros y criterios.

Una primera clasificación de los UAVs los divide en dos grandes grupos:

- **Vehículos de ala fija.** En este grupo se encuentran todo tipo de aviones.
- **Vehículos de ala rotativa.** En este grupo se encuentran tanto los helicópteros como todos los tipos de multicópteros.

Dentro de esta clasificación podemos encontrar un rango de vehículos que puede ir desde cometas, globos o misiles hasta aviones radios controlados o aeronaves prácticamente autónomas. Sin embargo, los tipos de UAV más comunes en la mayoría de estas aplicaciones son los cuadricópteros (Figura 2-2), ya que son capaces de transportar carga con la suficiente resistencia y estabilidad [1] [3].

Figura 2-2. Cuadricóptero.

También es común encontrar una primera clasificación de los UAV según las tareas que pueden realizar, atendiendo a las funcionalidades más comunes [1]:

- **Blanco**. Sirve para simular aeronaves enemigas o misiles en sistemas de defensa.
- **Reconocimiento**. Ofrece sistemas de inteligencia en el campo de batalla.
- **Combate**. Sirve para defender en misiones arriesgadas.
- **Logística**. Diseñados específicamente para realizar operaciones de logística.
- **Investigación y desarrollo**. Sirven para probar sistemas en desarrollo.
- **UAVs civiles y comerciales**. Diseñados exclusivamente con fines comerciales.

Para una clasificación más detallada se ha elegido la clasificación *NATO Joint Capability Group On Unmanned Aerial System* (*JCGUAS*). En esta clasificación se proponen 3 diferentes categorías de UAV dependiendo de su masa y una subdivisión de categorías de acuerdo a las siguientes características (Tabla 2-1).

- Propósito de la misión.
- Altura que pueden llegar a alcanzar.
- Distancia o duración en vuelo.
- Soporte en operaciones militares.

Weight classification group	Civil category	Wet mass (incl. fuel) kg	Regulation	Broad Military Equivalent	Notes and Systems
1	Molecular / femto satellites molesat	0.001 – 0.1	?	?	Launched to fly in an asterism pattern, known collectively as a constellation
1	Pico satellites picosat	0.1 – 1.0	?	?	Fly in formation as a swarm from mother satellite Black Widow MicroStar
1	Nano satellites nanosat	1 - 10	?	?	?
1	Small aircraft	0 - 20	National CAA	Micro (<20 kg) Mini (<30 kg)	?
2	Light UAV	>20 - <150	Civil	Tactical UAV Close range	Phantom, Mikado
2	Micro – small satellites	>10 - <100	Civil	Tactical UAV Short range (<200 kg)	Luna, Silver Fox, Firebird, Photo, Goldeneye
3	UAV	>150	EASA	Tactical UAV	
3	Mini – small satellites	>150 - <500	?	Short range (<200 kg) Medium range (150-500 kg)	Hunter, Aerostar, Sniper, Falco
3?	Special Task UAVs	250 250	Military	Lethal Decoys	MALI, Harpy, Lark, Marula Flyrt, MALD, Nulka, ITALD
4?	Tactical UAV	500 – 1,500	Military	Endurance range	Aerosonde, Vulture II Exp
4?	Tactical UAV	1,000 – 1,500	Military	Medium altitude long range MALE	Skyforce, Hermes 1500, Heron TP
4?	Strategic UAVs	2,500 – 12,500	Military	High altitude long endurance HALE	Global Hawk, Raptor, Condor, Theseus

Tabla 2-1. Clasificación JCGUAS.

Dentro de esta configuración en los últimos años han cobrado especial importancia los Micro-UAV, llegando a aparecer diseños parecidos a los de un insecto aéreo, son utilizados principalmente en espacios reducidos e incluso en zonas interiores mediante la incorporación de sistemas autónomos de vuelo capaces de sustituir la navegación por GPS mediante algoritmos de localización a través del mapa del entorno percibido y la información recibida de los sensores del UAV.

En la tabla 2-2 se muestra una descripción de las configuraciones más comunes de Micro-UAVs. Como se puede observar en la tabla cada configuración tiene sus ventajas y

desventajas. Dependiendo de la misión una configuración se puede adaptar mejor que otras. Por ejemplo, los dirigibles tienen una gran eficiencia energética, resultando claves para vuelos de larga duración donde no se requieran velocidades de vuelo altas, como por ejemplo en las tareas de vigilancia. Sin embargo, en este tipo de vehículos se aumenta considerablemente la carga del sistema y no resultan tan adecuados para transporte o como vehículos aéreos de combate [4].

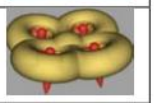
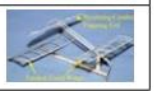
Configuration *e.g.*	Advantages	Drawbacks	Picture
Fixed-wing (AeroVironment)	Simple mechanics, silent operation	No hovering	
Single rotor (A. V de Rostyne)	Good controllability, good maneuverability	Complex mechanics, large rotor, long tail boom	
Axial rotor (Maryland Univ.)	Simple mechanics, compactness	Complex aerodynamics	
Coaxial rotors (EPSON)	Simple mechanics, compactness	Complex aerodynamics	
Tandem rotors (Heudiasyc)	Good controllability, simple aerodynamics	Complex mechanics, large size	
Quadrotor (EPFL-ETHZ)	Good maneuverability, simple mechanics, increased payload	High energy consumption, large size	
Blimp (EPFL)	Low power, long flight operation, auto-lift	Large size, weak maneuverability	
Hybrid quadrotor-blimp (MIT)	Good maneuverability, good survivability	Large size, weak maneuverability	
Bird-like (Caltech)	Good maneuverability, compactness	Complex mechanics, complex control	
Insect-like (UC Berkeley)	Good maneuverability, compactness	Complex mechanics, complex control	
Fish-like (US Naval Lab)	Multi-mode mobility, efficient aerodynamics	Complex control, weak maneuverability	

Tabla 2-2. Clasificación Micro-UAVs

2.1.2 Aplicaciones

En los últimos años el desarrollo de UAS se ha expandido significativamente, encontrando una gran variedad de tipos de UAV con un diseño específico dependiendo de la misión a realizar.

Históricamente los UAV han sido diseñados para realizar lo que se considera como *3D mission (Dull, Dirty and Dangerous),* minimizando el *payload* de acuerdo a la misión a desarrollar y sus necesidades específicas.

Un ejemplo de *Dull mission* podrían ser las misiones que requieren un vuelo de larga duración y tareas repetitivas como las de supervivencia o vigilancia. En estos casos, las limitaciones físico-psicológicas de un humano podrían afectar a los objetivos de la misión. Los UAS, en cambio, no presentan este tipo de problemas, ya que la persistencia en la misión sólo se ve afectada por el tiempo de vuelo del mismo, mientras que el operador se limita a supervisar el correcto funcionamiento del vehículo.

Las misiones consideradas como *Dirty* se caracterizan por la operación en entornos peligrosos para la salud humana, como podrían ser entornos con alto nivel de radiación, contaminación o compuestos químicos. Un ejemplo de estas misiones se puede encontrar en las recientes actividades monitorizadas por el *Global Hawk* sobre el reactor de Fukushima en 2011.

Finalmente, las misiones consideradas como peligrosas (*Dangerous)* se definen más en un contexto militar. En particular el propósito de estas es substituir los típicos bombarderos con vehículos aéreos tripulados o el reconocimiento de objetivos aéreo.

Dentro de este contexto, se puede encontrar la siguiente clasificación de aplicaciones dentro de un ámbito militar [1] [5]:

- Recolección inteligente de información:
    - Recolección inteligente de imágenes (IMINT).
    - Inteligencia de comunicaciones (COMINT).
    - Inteligencia electrónica (ELINT).
    - Inteligencia de señales (SIGINT).

- Supervivencia.
- Reconocimiento.
- Detección de artefactos explosivos.
- Asistencia en el campo de batalla.

Además de las aplicaciones militares, también se espera que los UAS sean utilizados para aplicaciones civiles. En la mayoría de ellas los UAS proporcionan asistencia a equipos o individuales:

- Seguridad:
  - Vigilancia en fronteras.
  - Control de leyes.
  - Control del contrabando.
  - Monitorización de grandes eventos y espectáculos.
- Monitorización de territorios dañados.
- Tareas de búsqueda.
- Asistencia en la industria agricultora.
  - Dispersión de fertilizantes.
  - Dispersión de pesticidas.
  - Monitorización del cultivo.
- Asistencia en el sector pesquero.
- Control medioambiental e investigación meteorológica.
- Explotación de mineral.
- Monitorización de costas.
- Detección/ Monitorización de áreas contaminadas.
- Transmisión de redes de telecomunicación.
- Retransmisión de telenoticias.
- Control de tráfico aéreo.
- Control de tráfico terrestre.
- Control de tráfico marítimo.
- Reproducción de mapas terrestres.
- Asistencia en incendios.
- Monitorización de canales.

2.1.3 Operadores de UAVs

En este capítulo se hace una revisión de los posibles operadores de vehículos aéreos no tripulados. El tipo de operador resulta un factor clave en la identificación de usuarios de la aplicación y cada uno de ellos introduce unas necesidades especiales y nivel de dependencia con el vehículo y en particular con la herramienta.

Actualmente no hay una clara posición del operador con el vehículo aéreo no tripulado, llegando a adoptar cada uno posiciones diferentes dependiendo de tipo de sistema a bordo del vehículo o la propia cultura, influencia o experiencia del operador. Considerando un UAS de clase III y las especificaciones de interoperabilidad que se proponen en el estándar STANAG 4586 [2] [5], se podrían identificar los siguientes posibles operadores.

- **Operador del vehículo**. es el responsable del control del vehículo, a veces suele coincidir con el responsable de carga.
- **Responsable de la carga**. Es el responsable de controlar el cargamento a bordo del vehículo, en caso de que lo halla.
- **Comandante de la misión**. Es el encargado de dirigir la misión y el responsable final del vehículo aéreo.
- **Operador analista de datos.** Este operador está destinado al análisis de datos o imágenes en tiempo real y su posterior explotación.
- **Especialista de comunicaciones.** Operador dedicado a la comunicación con otros agentes operativos en el escenario.

Además de esta clasificación, es importante tener en cuenta la experiencia del operador en el diseño de la interfaz. Para el caso de un operador con experiencia, por ejemplo se hace indispensable el uso de la simbología y controles que suelen formar una cabina de mando. Sin embargo, para un operador más genérico se podrían utilizar otro tipo de métodos de visualización adaptados similares a los distintos tipos de interfaces que se ofrecen en otras aplicaciones informáticas. En cualquier caso, independientemente del tipo de operador y vehículo hay una complejidad añadida entorno al nivel de dependencia con vehículo o los distintos factores humanos debidos a la separación física existente entre el operador y el vehículo.

2.2 Niveles de autonomía

Los niveles de autonomía de cualquier sistema, suponen un factor determinante en el diseño de interfaces persona-ordenador, influyendo directamente en la cantidad de información que tiene que ser presentada al usuario. Así mismo, determinan los posibles modos de interacción u operación del usuario con el vehículo aéreo no tripulado, influyendo directamente en el nivel de complejidad de la misma. Por este motivo, en el capítulo siguiente, se hace un análisis de los diferentes niveles de autonomía en los vehículos aéreos no tripulados.

Los niveles de autonomía describen el grado en el que un sistema robótico es capaz de tomar decisiones y ejecutar diferentes acciones. Estos grados de autonomía hacen una diferenciación entre los sistemas que se pueden encontrar, clasificándolos en sistemas automáticos, sistemas autónomos y sistemas inteligentes.

- **Los sistemas automáticos** son sistemas encargados de realizar tareas pre-programadas sin ninguna capacidad de razonamiento o decisión.

- **Los sistemas autónomos** tienen la habilidad para sentir, percibir, analizar, comunicar, planificar, tomar decisiones y actuar para lograr conseguir los objetivos asignados por un operador humano a través del diseño del mismo. Este nivel de operación involucra otros procesos relacionados como podría ser la automatización definida como la pre-programación de actuación como respuesta a un estímulo específico, o la adaptabilidad referida a la capacidad de reacción a cambios en el entorno sin la asistencia de un operador.

- **Los sistemas inteligentes** se definen como sistemas autónomos con la capacidad de decidir o generar sus propias metas u objetivos dependiendo de sus propias motivaciones.

2.2.1 Niveles de autonomía ALFURS

En 1192 Tom Sheridan [6] propuso 10 niveles de autonomía, partiendo de sistemas robóticos controlados casi completamente por el ser humano hasta sistemas robóticos completamente autónomos, los cuales no requieren ningún tipo de interacción humana. Dentro de estos 10 niveles se describen los siguientes:

1. El ordenador no ofrece asistencia, el humano hace todo.
2. El ordenador ofrece una serie de acciones alternativas.
3. El ordenador estrecha la selección a unas pocas elecciones.
4. El ordenador sugiere una única acción
5. El ordenador ejecuta la acción si el humano la aprueba.
6. El ordenador permite al humano un tiempo de aceptación de la acción antes de ser ejecutada automáticamente.
7. El ordenador ejecuta automáticamente la acción informando al humano.
8. El ordenador informa al humano después de ejecutar la acción automáticamente, solo si el humano pregunta.
9. El ordenador informa al humano después de ejecutar la acción automáticamente, solo si el ordenador decide que debe hacerlo.
10. El ordenador decide todo y actúa automáticamente ignorando al humano.

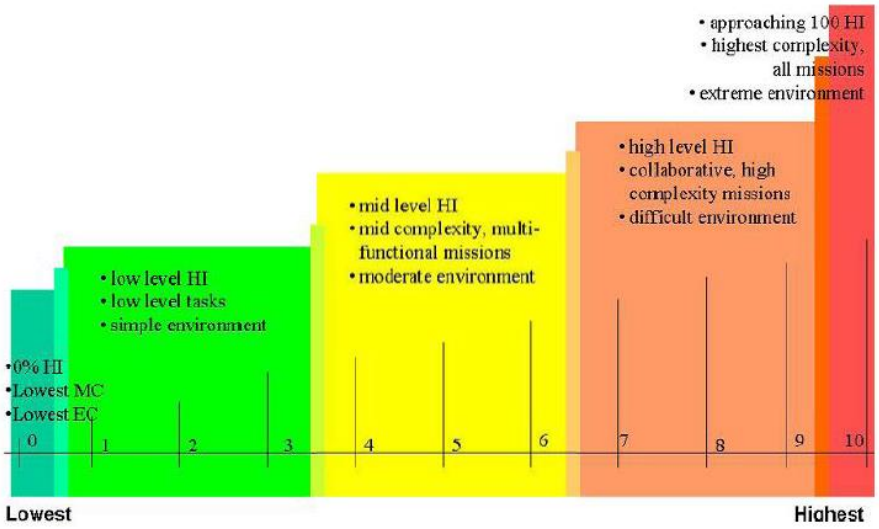


Figura 2-3. Clasificación de Tom Sheridan.

Sin embargo, este método de clasificación no puede ser aplicado a cualquier dominio, ya que resulta muy complicado establecer una cuantificación de los niveles a niveles que pueda ser aplicado en cualquier contexto. Por este motivo, en el año 2000 se propuso una revisión usando modelos divididos en 4 clases de funciones [7]. Estas clases de funciones son la adquisición de la información, análisis de la información, selección de decisiones y acciones y la implementación de las acciones.

En 2002, el *US Air Force Research Laboratory* (AFRL) realizó una nueva revisión, presentando lo que se conoce como "*Autonomous Control Level (ACL)"* para medir los diferentes niveles de autonomía en los vehículos aéreos no tripulados. Este modelo utiliza una tabla de 11 niveles de autonomía basado en OODA (*Observe, Orient, Decide and Act*) que hace una división de 4 clases de acciones principales ejecutadas por los UAV.

La observación está relacionada con la percepción y el conocimiento, la orientación con el análisis y coordinación, la decisión con las elecciones y por último la acción con la habilidad para ejecutar acciones.

En 2007, el *National Institute of Standards and Technology (NIST)* desarrolló un marco en el que se definen los diferentes niveles autonomía para sistemas autónomos. Este Marco se denominó *Autonomy Levels For Unmanned Systems (ALFUS).* Este marco utiliza diferentes métricas, evitando significativas diferencias en las transiciones existentes entre distintos niveles.

Para esta clasificación se establecen tres métricas determinantes, *Human Independence (HI), Mission Complexity (MC)* y *Environmental Complexity (EC).* Cada una de estas métricas define varios factores. Por ejemplo, la complejidad del entono (EC) podría

depender del terreno y otros factores meteorológicos, además de la complejidad derivada de la diversidad de objetos del escenario.

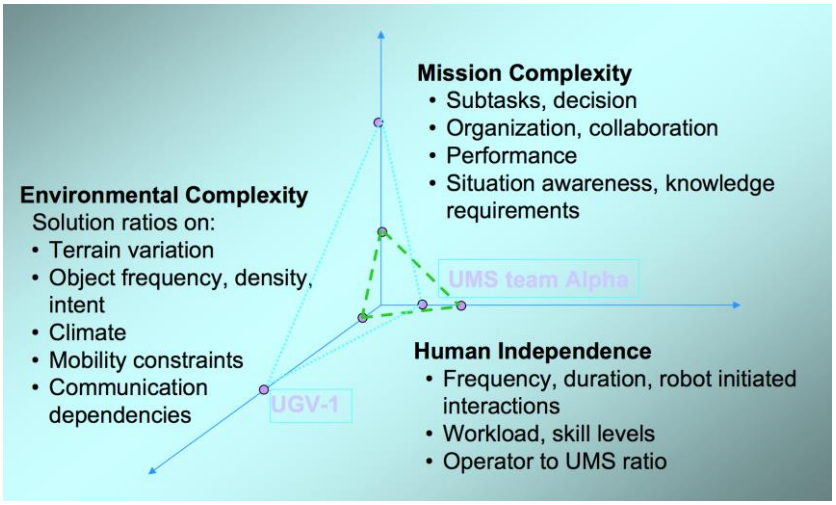


Figure 2-4: ALFUS Métricas de clasificación.

Actualmente el marco ACL es usado principalmente en UAVs de grandes dimensiones que pueden soportar vuelos a grandes alturas y libres de obstáculos. Aplicándose el marco ALFUS en un contexto general junto con otros marcos diseñados para un tipo particular de UAV. En 2012, *Kendoul* [8] propuso el *Rotorcraft Unmanned Aerial System (RUAS)* que realiza una clasificación específica para helicópteros, obteniendo un nuevo marco conocido como *Autonomy Levels For Unmanned Rotorcraft Systems (ALFURS)* [4].

2.3 Modos de Vuelo

En la práctica, no existe un estándar que establezca unos modos de operación y haga distinción entre los distintos niveles de autonomía. Sin embargo, cada modo de control impone necesidades distintas al operador. En el siguiente capítulo, se hace una revisión de los modos de operación para sistemas automáticos y autónomos, revisando las características de cada uno de ellos y las necesidades específicas del operador.

Actualmente, un vehículo aéreo no tripulado puede ser controlado de maneras diferentes mediante el modo de vuelo seleccionado, además se sabe de la existencia de modos híbridos como, por ejemplo, modos guiados automáticos que siguen una ruta horizontal en el plano, mientras que la altitud y la velocidad son controladas mediante un modo semiautomático dirigido por el operador. Un vehículo aéreo no tripulado con la capacidad de funcionar de manera autónoma se podría decir que se comporta de la misma manera que un vehículo automático desde el punto de vista del control del vehículo, sin embargo, ambos comportamientos deben ser distinguidos ya que implican niveles diferentes de decisión en el usuario y sistema.

En una primera división de los modos de operación con vehículo aéreos no tripulados se podría diferenciar el control externo del vehículo *Direct Line of Sight* frente al control interno *Inside Control Station* a través de un piloto de vuelo [5]. En esta clasificación se puede encontrar una gran diferencia en cuanto al nivel y la forma de presentación de información por la HMI. Por ejemplo, en un control externo se podría encontrar un escenario en el que se está conduciendo el vehículo remotamente, si el vehículo se aleja del operador y el operador envía un comando para moverlo hacia la derecha el vehículo se moverá hacia la derecha, sin embargo si el vehículo se acerca al operador se moverá hacia la izquierda respecto a la posición del operador.

2.3.1 Sistemas automáticos

Este tipo de sistemas tienen un nivel de dependencia mayor con el usuario u operador de la interfaz, resultando clave proporcionar el tipo de interfaz adecuado que ayude al usuario. A continuación se hace una descripción de los tipos de control más comunes en los sistemas automáticos, atendiendo al nivel de dependencia con el operador y el tipo de interfaz que se suele proporcionar en cada caso [5].

- **Control automático.** El UAV sigue una ruta usando métodos avanzados de navegación como podría ser navegación por GPS. El operador es capaz de cambiar uno o varios atributos de la ruta. El tipo de interfaz que se proporciona en este caso es un mapa que muestra la ruta que ha de seguir el vehículo.

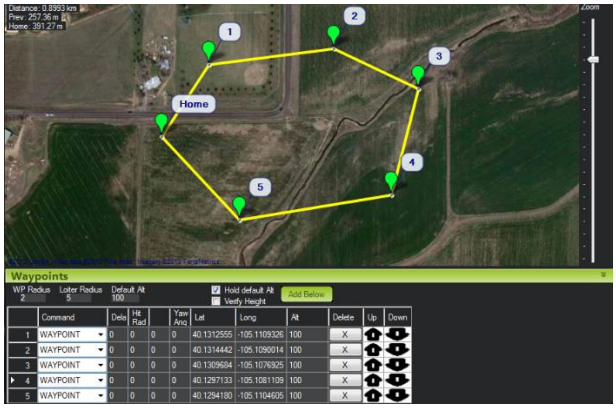


Figura 2-5: Interfaz Control Automático

- **Control semiautomático.** El UAV es capaz de seguir valor discretos tales como la altitud, velocidad, velocidad vertical o el valor de cabeceo proporcionados por el operador. El modo de guiado, en este caso, es calculado por el sistema autopiloto a bordo del vehículo. La interfaz que se suele proporcionar en este

modo está basada en la visualización de un panel que muestra y permite interaccionar con los principales valores del sistema autopiloto.

Figura 2-6: Interfaz Control Semiautomático

- **Control manual.** El UAV está dirigido remotamente por el operador usando comandos de control de vuelo convencionales como podría, por ejemplo, ser el empuje para despegar o aterrizar. Este tipo de control es de lazo abierto, ya que es el sistema autopiloto a bordo del vehículo el que calcula los valores de salida para los motores a partir de los controles *roll*, *yaw*, *throttle* y *pitch* que recibe del usuario. Sin embargo, también se podría distinguir un tipo de control de lazo cerrado, que se suele denominar estabilizado, en este caso el controlador de orientación calcula los valores de salida para los motores a partir de los controles que da el usuario y la retroalimentación de la situación actual. Los controles se manejan por tanto de manera manual, pero el vehículo se estabiliza automáticamente.

Figura 2-7: Interfaz Control Manual

2.3.2 Sistemas autónomos

Este tipo de sistemas tienen un nivel de dependencia menor con el usuario u operador de la interfaz, sin embargo, se debe informar al usuario correctamente del estado del vehículo ya que éste es el responsable final de sistema. En este tipo de interfaces juega un papel importante la visualización de la visión del entorno que tiene el vehículo aéreo, ya que es el que le va a permitir responder frente a él y llevar a cabo la misión de forma satisfactoria.

En este tipo de sistemas además de los mencionados anteriormente se pueden encontrar dos nuevos modos de control, el modo de vuelo semiautónomo y el modo de vuelo autónomo (Tabla 2-3).

	No autónomo	**Semiautónomo**	**Autónomo**
Trayectoria de Vuelo	Preprogramada	Parcialmente autónomo. Posibilidad de recalcular la ruta	Decisión sobre la trayectoria y cálculo de la ruta
Presencia del piloto	Guiado y control continuo	Supervisión continua. Guiado y control parcial.	Sólo como respaldo
Necesidad de actuación del piloto	En todos los casos.	Posible replanificación de la ruta por parte del operador.	Sólo como respaldo

Tabla 2-3. Clasificación según el grado de autonomía.

En el modo de vuelo semiautónomo, el operador carga una serie de *waypoints* pudiendo realizar cambios durante el vuelo. El operador entonces se limita a monitorizar la trayectoria del vehículo y realizar correcciones.

En el modo de vuelo autónomo, se espera una operación inteligente por parte del vehículo para llevar a cabo la misión. Mientras que el operador se limita a monitorizar el correcto funcionamiento del sistema y actuar en caso de fallo en el mismo.

2.4 Interfaces para vehículos aéreos no tripulados

Normalmente las interfaces gráficas se diseñan de acuerdo a unas directrices generales definidas previamente y que sirven como punto de partida en el diseño general de la aplicación. Algunas de estas directrices son comunes para cualquier tipo de aplicación, estando incluso los términos que identifican a los componentes comúnmente aceptados en la literatura.

Sin embargo, la gran mayoría derivan de estos componentes básicos y forman parte del contexto específico de la aplicación, no existiendo una especie de *checklists* que garantice un buen diseño de una interfaz.

Algunas de estas heurísticas que han de ser previamente consideradas en el diseño son [23]:

- Mostrar la información de forma ordenada y consistente, evitando reformulaciones y/o transposiciones por el operador que puedan aumentar el volumen de información que se presenta.
- Evitar tener diferentes tipos de formatos para representar la información, reduciendo de esta manera la habilidad nemotécnica del usuario.
- Usar un lenguaje simple y natural que esté presente en la literatura.
- Proporcionar retroalimentación de los comandos y el estado general de la aplicación.
- Evitar errores por parte del usuario, pidiendo confirmaciones de las acciones relevantes.
- Optimizar la visibilidad y legibilidad, evitando tener elemento en segundo plano que puedan hacer interferencia o distraer al usuario.
- Presentar solo la información que es útil para el usuario en un momento dado.
- Implicar directamente al usuario de la aplicación en el diseño de la misma.
- Usar una disposición simétrica en forma de rejilla.
- Estandarizar el estilo de la ventana y los elementos básicos.
- Utilizar colores para informar y no para decorar la aplicación.

Para el diseño de los componentes específicos de la interfaz se ha hecho una revisión de las diferentes interfaces existentes en el mercado. Finalmente, se ha decidido tomar como punto de partida la aplicación *QgroundControl*, ya que está ampliamente utilizada y aceptada por la comunidad.

Sin embargo, hay que destacar que esta aplicación está orientada más a sistemas automáticos, ofreciendo características y funcionalidades específicas a estos sistemas y a los operadores de los mismos, que no pueden ser llevados al diseño de nuestra aplicación. En el apartado de diseño 4 se describe con mayor detalle el diseño de la arquitectura para un sistema autónomo y se hace una revisión de las funcionalidades básicas que se considera que debe ofrecer la aplicación respecto a este tipo de sistema.

2.4.1 Ejemplos de interfaces de usuario

QGroundControl es una aplicación *opensource* destinada a pilotar mediante puntos de coordenadas (*waypoints*) aviones y helicópteros en sistemas con pilotaje automático, llegando a soportar la gran mayoría de los sistemas autopilotos existentes en el mercado como el *arduPilotMega* o el *pxIMU* y pudiendo ser utilizado con la mayoría de los drones comerciales como podría ser el *AR.Drone* con su *GPS* o con *flight recorder* [9].

En este sentido esta aplicación permite visualizar y controlar el vehículo aéreo no tripulado durante el desarrollo y operación de una misión.

Principales características:

- Utiliza como protocolo de comunicaciones el Open Source MAVlink (*Micro Air Vehicle Communication*)
- Ofrece mapas aéreos 2D/3D con *drag* and *drop waypoint* (arrastrar y soltar).
- Cambio y manipulación de parámetros de vuelo de la electrónica de control en tiempo real, incluso con el vehículo en vuelo.
- Visualización en tiempo real de datos de sensores y telemetría.
- Es multiplataforma.

Figura 2-4: QgroundControl

En la figura 2-4 se puede observar una de las vistas principales que ofrece esta interfaz con el mapa aéreo 3D y dos sistemas de visualización de la posición, altitud y orientación del vehículo en la parte derecha de la figura.

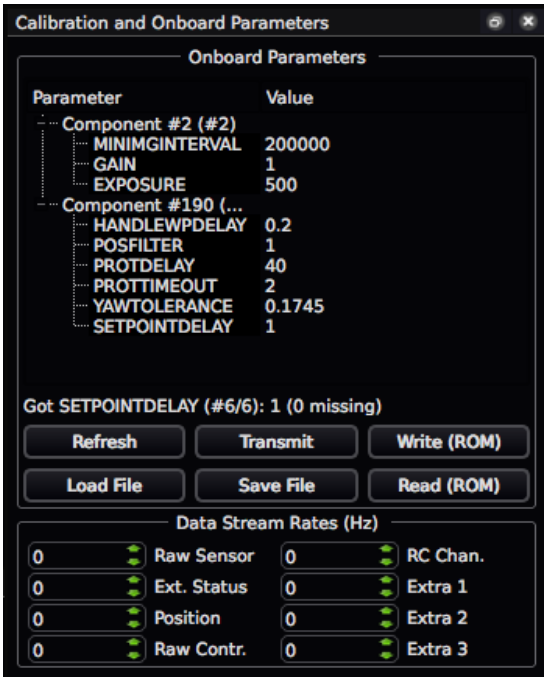


Figura 2-5: Reconfiguración de parámetros

Además de ofrecer múltiples sistemas visualización del entorno en 3D, permite la reconfiguración de los parámetros de vuelo, ofreciendo al usuario una interfaz gráfica flexible que abstrae de la complejidad de la arquitectura de control del vehículo y fomenta la interoperabilidad de la herramienta con los diferentes componentes del sistema en vuelo.

Otra aplicación que ha servido de punto de partida ha sido el sistema *HappyKillmore*. Está aplicación también es *opernsource* y acepta la mayoría de los sistemas autopilotos. Sin embargo al contrario que *QGroundControl*, está aplicación ofrece un diseño minimalista que simplifica bastante la interacción del usuario con la aplicación, ofreciendo los componentes básicos para la operación y supervisión con el vehículo.

Como se puede observar este sistema también incluye un mapa 3D de navegación junto con una vista parecida a una cabina de mando de un piloto de avión. Además de los relojes como indicadores de la posición, altitud, velocidad y orientación se proporciona una modelo de vehículo 3D a través de la cual se informa al usuario de forma visual de la dinámica general del vehículo.

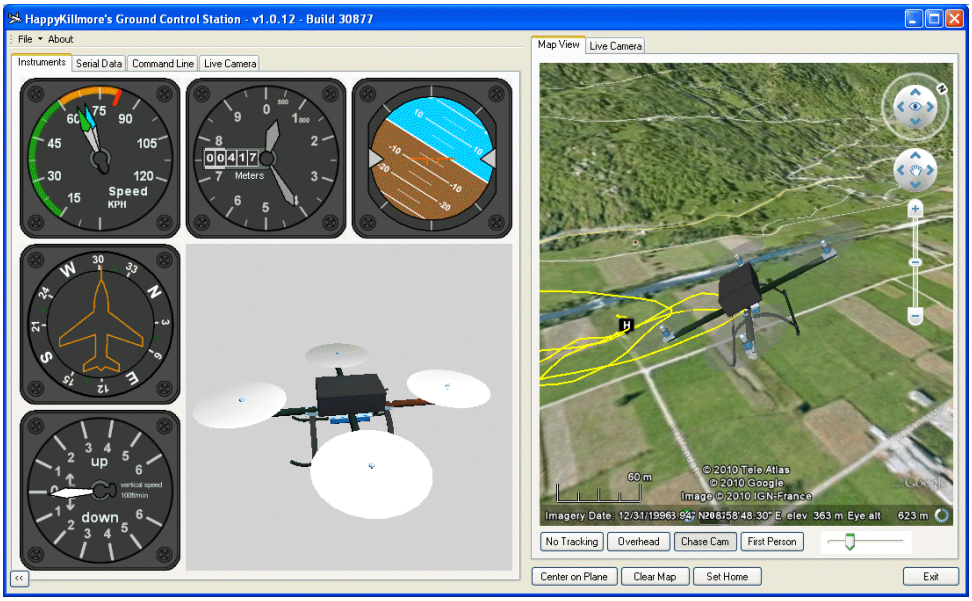


Figura 2-6: HappyKillmore's

2.4.2 Análisis de componentes

A continuación se hace una revisión de las características funcionales y del diseño de cada uno de ellos, describiendo las ventajas e inconvenientes que ofrece cada uno desde el contexto donde se desarrolla la interfaz, objeto de este documento.

QGroundControl

Dentro de las ventajas funcionales que soporta *QGroundControl* se podría mencionar el uso de una interfaz reconfigurable para diversos tipos de sistemas autopilotos a través de la propia interfaz de usuario que ofrece la aplicación y el uso del protocolo de comunicaciones *MAVLink*. Este protocolo de comunicación ofrece un sistema flexible y ligero, ofreciendo librerías de código abierto que permiten trabajar con datos, variables y estructuras típicas del lenguaje "C" para diversos *microcontroladores* como LPC, AVR vía radio módem.

Como inconvenientes se podría mencionar la dificultad de uso de la aplicación para un usuario medio. Esto es debido a la gran cantidad de funcionalidades que ofrece como la posibilidad de reconfigurar los parámetros de la electrónica de control y el hecho de que el diseño de la interfaz se pueda prácticamente reconfigurar en su totalidad, obteniendo a veces un diseño complejo que complica la interacción del usuario con la aplicación.

HappyKillmore's

La aplicación *HappyKillmore's*, sin embargo, en este sentido ofrece un diseño minimalista fácil de utilizar. Además divide la ventana principal en dos zonas bien diferenciadas a través de las cuales se ofrece por un lado una vista general con mapa 3D o las propias imágenes recibidas de las cámaras del vehículo y, por el otro lado, una vista con el detalle de la información de vuelo o la dinámica del vehículo. La mayoría de las configuraciones necesarias para el funcionamiento de la aplicación con el vehículo se realizan automáticamente, dejando sólo al usuario la selección de puerto y la tasa de baudios.

3 DISEÑO

Este capítulo describe el diseño realizado en forma de herramienta de interacción persona-ordenador para la operación de vehículos aéreos no tripulados. A esta herramienta se le ha llamado HMI (Human Machine Interface), uno de los componentes de la arquitectura software del UAV.

Para ello, primero se hace una descripción de la arquitectura a bordo del vehículo y se realiza una división de funcionalidades entre cada uno de los componentes del sistema autónomo y la HMI. Finalmente, se establece el grado de interacción con la interfaz a través de factores humanos que resultan claves en el diseño como podrían ser la consciencia situacional o la acción supervisora que realiza el usuario a través del HMI.

A través de este análisis se especifican los requisitos funcionales de la herramienta y se describe el diseño realizado de cada uno de los componentes que la integran.

3.1 Visión general de la arquitectura

En el diseño de una interfaz de usuario para sistemas autónomos es importante tener en cuenta por separado cada uno de los elementos del sistema con el que va a operar la interfaz. Normalmente la complejidad de estos sistemas obliga a que estén diseñados por un equipo numeroso de personas cada uno especializado en un campo específico como podría ser el campo de la visión computacional, los sistemas de control o la propia inteligencia artificial del vehículo.

Sin embargo, validar todos estos componentes que cada uno ha ido validando por separado en una plataforma de vuelo real resulta una tarea bastante compleja si no se cuenta con una interfaz que permita al usuario comunicarse con el vehículo aéreo.

Este tipo de comunicación entre el vehículo y el operador además de ofrecer una visión de la arquitectura en general y del funcionamiento del sistema en vuelo, también amplia las capacidades del usuario de la interfaz ofreciéndole una visión del entorno o permitiéndole operar en zonas que antes eran inaccesibles.

Por este motivo, en este apartado se hace un estudio de la arquitectura para la cual se ha diseñado esta interfaz, atendiendo a cada uno de los componentes y realizando una correspondencia de funcionalidades que le permitan al usuario comunicarse con el vehículo aéreo de manera eficaz. En este estudio, además de tener en cuenta cada uno de los componentes de la arquitectura, también se ha tenido en cuenta la respuesta que el

usuario debe ofrecer a cada uno de ellos y la interacción hombre-máquina que tiene lugar entre cada uno de estos componentes y la herramienta.

3.1.1 Arquitectura de un vehículo aéreo no tripulado autónomo.

Los sistemas software que gestiona un vehículo aéreo no tripulado deben ofrecer una serie de características y funcionalidades determinadas, tales como los sistemas de percepción o de control del UAV. Sin entrar en detalles sobre las diferentes arquitecturas existentes en general, se podría decir que una buena arquitectura bien dividida y clasificada ofrece un diseño eficiente y modular, entre otras cosas, al sistema que lo utiliza.

En particular, el HMI se puede ver como un componente más de la arquitectura general ofreciendo la funcionalidad de comunicación entre el usuario y cada uno de los subsistemas a bordo del vehículo. Sin embargo, la complejidad del diseño de las arquitecturas autónomas y la falta de uniformidad en cuanto a diseño y funcionalidades que estas pueden llegar a ofrecer en un sistema real, ha llevado a tener en cuenta el diseño del HMI como un componente autónomo y modular que se puede conectar a la arquitectura siempre que ésta admita unos puntos esenciales y funcione sobre el middleware ROS descrito en capítulos posteriores.

Para ilustrar e identificar los diferentes subsistemas o componentes con los que se debería de comunicar la interfaz se ha realizado un diseño de alto nivel de la arquitectura software abstrayendo la complejidad de implementación y atendiendo a las funcionalidades que cada uno de ellos ofrece al sistema en general.

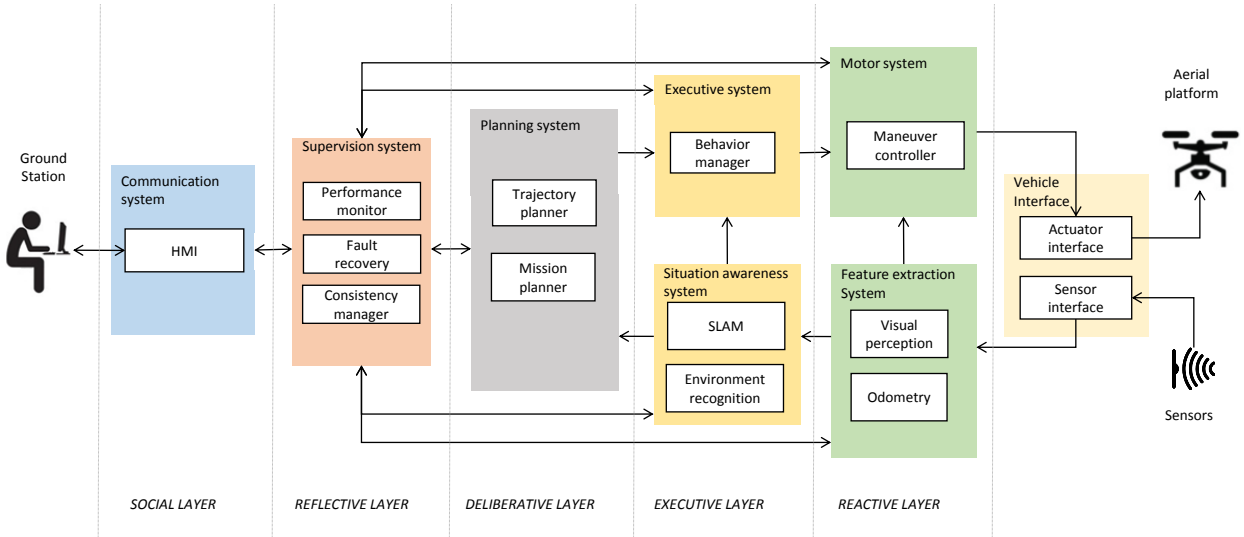


Figura 3-1: Componentes principales de la arquitectura software.

El diseño general de esta arquitectura se ha construido sobre el sistema software *cvg_quadrotor_swarm* [10] desarrollado por el CAR y que ha ganado varios premios en distintas competiciones (IMAV & IARC) demostrando así su capacidad en entornos de vuelo complejos. Se ha elegido ha elegido este sistema software frente a otros por ser un sistema robusto que ha sido comprobado y utilizado en un sistema de vuelo real.

3.1.2 Estructura general de la arquitectura

Para el diseño de la arquitectura de alto nivel se han tenido en cuenta los diferentes paradigmas de la robótica inteligente. Un paradigma es definido por Murphy [11] como "*una filosofía o conjunto de asunciones y/o técnicas que caracterizan una aproximación a una clase de problemas.".* En concreto existen 3 paradigmas básicos en la robótica inteligente basados en las 3 funcionalidades básicas divididas en sentir, planificar y actuar y diseñados de acuerdo a como se distribuye la información percibida a través del sistema:

- **Paradigma Jerárquico o deliberativo**. Este paradigma se basa en la manera de pensar introspectiva de las personas. Bajo este paradigma el robot comienza percibiendo el mundo, después planifica una acción de acuerdo a la abstracción del mundo obtenida y ejecuta una acción.

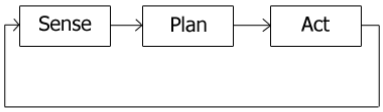


Figura 3-2. Paradigma Jerárquico.

- **Paradigma Reactivo.** Este paradigma tiene menor coste computacional que el paradigma jerárquico, diseñado para ofrecer una respuesta o acción sobre el medio percibido en el menor tiempo posible eliminando la planificación de la acción.

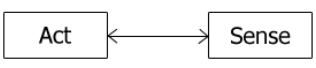


Figura 3-3. Paradigma Reactivo.

- **Paradigma Híbrido.** Es el más usado actualmente ya que resuelve la falta de toma de decisiones del paradigma reactivo y el tiempo de ejecución del paradigma deliberativo. Para ello se ha aprovechado el tiempo que tarda el planificador en tomar una decisión ejecutando comportamientos reactivos en paralelo, quedando el siguiente esquema:

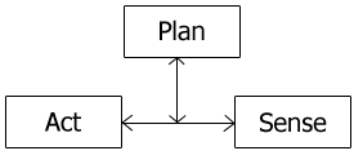


Figura 3-4. Paradigma Híbrido.

Siguiendo estos paradigmas y otros utilizados [12] [13] [14] se han diseñado además de la capa reactiva y deliberativa las siguientes capas complementarias:

- **Capa ejecutiva**. Memoriza el estado pasado como una representación modelo del mundo obtenido de la información recibida por los sensores y acepta directivas de la capa deliberativa para secuenciarlas y enviárselas a la capa reactiva.

- **Capa refractiva.** Tiene en cuenta los comportamientos que no son tratados por el resto de componentes del sistema y ofrece una respuesta a estos comportamientos.

- **Capa social.** Este paradigma ofrece una comunicación bidireccional entre la arquitectura y los diferentes agentes existentes en el entorno del robot.

3.1.2.1 Vehicle interface

Se puede figurar fácilmente cual es la tarea que desempeña este módulo al ser la conexión directa con la parte más física, la plataforma aérea y el resto del hardware como los sensores. Este sistema ofrece una interfaz al resto de la arquitectura para gestionar el UAV, reduciendo la complejidad de tratar con las partes físicas como el motor.

Tenemos dos módulos o subsistemas que se comunican con los sensores y el controlador de bajo nivel:

- **La interfaz de actuación** (*actuator interface*) que recibe comandos de la arquitectura como *Thrust, Pitch* o *Roll* e interacciona directamente con el motor produciendo el cambio deseado.

- **La interfaz de sensores** (sensor interface) recibe la información obtenida de las cámaras y de otros sensores como el LIDAR ofreciendo un filtrado muy básico y simple.

3.1.2.2 Motor System

Este sistema permite transformar cualquier plataforma aérea en una plataforma genérica y fácil de manipular. Esto nos permite abstraer el hardware de la plataforma aérea del resto de la arquitectura, facilitando el trabajo de más alto nivel y trabajando de forma independiente.

El uso de una plataforma aérea u otra, repercutirá en usar unos ficheros de configuración u otros en el Motor System. De esta forma, al recibir un comando de alto nivel como seguir a un objetivo el UAV lo realizará de manera independiente de la plataforma aérea usada, siempre que ésta haya sido analizada previamente.

3.1.2.3 Feature Extraction System

En este módulo se aplica un primer nivel de comprensión del exterior cómo podría ser la identificación de los distintos objetos del entorno, apoyándose en módulos y librerías ya existentes como podría ser el *Aruco Eye*. A través del sistema de sensores se puede obtener información del propio robot y del entorno que le rodea. Éstos pueden ser muy diversos, sensores giroscópicos, acelerómetros, sensores de presión, cámaras, sistemas de flujo óptico, todos ellos proporcionan información en bruto para su análisis en los siguientes subsistemas.

3.1.2.4 Situation Awareness System

El UAV dispone de diversos mecanismos para percibir información acerca del mundo que le rodea a través del sistema de sensores a bordo del vehículo. Sin embargo, el ruido presente en los sistemas sensoriales hace imposible construir modelos perfectos del entorno, que en la mayoría de los casos suele ser dinámico y confuso generando progresivamente crecientes cantidades de información.

Por este motivo, es necesario realizar un análisis de la información que permita construir un modelo aproximado del entorno, llegando a obtener mapas del entorno y/o la posición virtual sobre la que se encuentra el vehículo aéreo en dicho mapa para que su contexto, movimiento y acciones puedan ser procesadas por otros sistemas.

En este sentido, se pueden diferenciar dos subsistemas principales involucrados en el proceso:

- Estimación de estado y posicionamiento (*SLAM*): Es la capacidad que tiene el sistema para estimar las variables relativas al posicionamiento y movimiento del UAV como la altitud, posición y velocidad, mediante el procesamiento de las medidas recibidas por los sensores.

- Reconocimiento del entorno (*Enviroment recognition*): Es la habilidad para construir un modelo interno del entorno a través de las entradas generadas por los sensores. La percepción se puede dividir en diferentes niveles o funciones como podrían ser mapeo, detección de objetivos y obstáculos, reconocimiento de obstáculos etc.

3.1.2.5 Executive system

Este sistema se encuentra en un punto intermedio entre el Motor System y el Planning System. Su función es descomponer las órdenes de la arquitectura software en partes más sencillas y entendibles por el Motor System activando y desactivando controladores que afecten a dicho sistema.

3.1.2.6 Supervision System

Como podemos observar con los subsistemas anteriores, el vehículo podría funcionar correctamente, aun así es necesario darle una capa de robustez para hacerlo autónomo. Esta robustez se puede lograr de dos maneras, de hecho, este sistema diferencia claramente dos subsistemas, uno en la vida de los procesos y otro en su comportamiento.

Mientras que la inclusión del segundo subsistema es discutible, el primero es necesario debido a que los procesos deben estar vigilados. Hipotéticamente, si un proceso falla por un error previsto o imprevisto se ha de tomar una decisión que no ponga en riesgo, la arquitectura, el UAV o, peor aún, su entorno.

Adicionalmente este módulo ha de registrar la información interna del vehículo para que pueda ser accesible si es requerida más adelante y filtrar la información que pueda ser relevante para mostrar a otros sistemas o usuarios.

3.1.2.7 Planning System

Para lograr un sistema autónomo e independiente, es necesario dotarlo de una mínima inteligencia para que el UAV pueda manejarse en el entorno al que es destinado y se comporte de una forma u otra según en la situación que se encuentre.

Para poder elaborar los comportamientos necesarios el UAV requiere saber cómo se encuentra, dónde se encuentra y qué ocurre a su alrededor, este punto ya está resuelto por los sistemas anteriores. Por lo tanto este sistema dispone de toda la información necesaria para que, a través de una serie de reglas, sistemas estadísticos, actuadores, etc. pueda mandar la respuesta u orden apropiada al UAV de la misma manera que el cerebro lo hace cuando tiene que mandar levantar la mano y agarrar un soporte o barra para no caer.

La función de este sistema, por tanto es seleccionar las acciones dentro de un conjunto de posibles escenarios tras procesar la información percibida, sustituyendo entonces las decisiones que podría llevar a cabo el operador del vehículo, formando el sistema cognitivo del robot.

A diferencia del resto de componentes de la arquitectura este componente se diseña específicamente para la misión en particular y los objetivos que esta propone. En este caso, este sistema se ha diseñado para la 7 misión del IARC que se describe en el capítulo siguiente.

3.1.2.8 Communication System

Este sistema representa a la interfaz persona-ordenador del sistema a bordo del UAV, objeto de esta memoria. La cual es vital cuando el operador o usuario requiera información acerca del UAV o del sistema abordo.

3.2 Análisis de los requisitos funcionales

La mayoría de las funcionalidades que son ejecutadas por un sistema de vuelo manual actualmente pueden ser ejecutadas desde los UASs a bajo nivel. Sin embargo, esta forma de ejecución autónoma actualmente ofrece un nivel menor de control, integración y respuesta respecto a las funcionalidades que tradicionalmente ofrece un sistema de vuelo manual. Por consiguiente, la inclusión de un sistema de vuelo manual y de un HMI es fundamental, no sólo para incrementar las capacidades de los vehículos aéreos no tripulados, sino para hacer posible la integración de estos sistemas en aplicaciones civiles, sin poner en riesgo la vida de las personas.

En este sentido, es posible establecer una relación entre cada una de las funcionalidades que ofrece un UAS y las ofrecidas por el HMI a través del sistema de control terrestre, de esta manera las funcionalidades del HMI requieren frecuentes interacciones con el operador de la estación de control terrestre perteneciendo a un bucle de control de alto nivel. En el vehículo aéreo no tripulado, en cambio, todas las funcionalidades se localizan dentro de un bucle de control interno como podrían ser las funcionalidades de navegación, guiado, etc. Además de las funcionalidades comunes, los UAS autónomos suelen contar con funcionalidades que no son controladas por el sistema de vuelo manual de la HMI como podría ser la re planificación autónoma del vehículo aéreo.

En general, es sencillo considerar todas aquellas funcionalidades correspondientes a las que suele ofrecer un vehículo aéreo tripulado, existiendo siempre una equivalencia en cada una de ellas, sin embargo, ésto no ocurre cuando se consideran funcionalidades más complejas especialmente las relativas a la planificación de la misión y la autonomía del vehículo. De este modo se ha realizado un análisis y se ha establecido una división y correspondencia entre las funcionalidades de cada sistema empezando por las funcionalidades más comunes, considerándose como UAV de referencia uno de Clase III de tipo MALE. El hardware que incluye el vehículo, sin embargo, se ha considerado irrelevante en este contexto ya que no actúa directamente con el HMI.

Funcionalidad	HMI	Vehículo aéreo no tripulado
Planificador de trayectorias	Ofrece un visualizador de trayectoria y la visualización de la dinámica del vehículo. Permite la redefinición de la trayectoria como una secuencia de waypoints definidos por el usuario.	Dado un objetivo, calcula la trayectoria a seguir para cumplir este objetivo de manera satisfactoria.
Sistema de Percepción	Ofrece una visualización del conocimiento que tiene el vehículo del entorno.	Obtiene una abstracción del entorno a través de los diferentes algoritmos de percepción que implementa el sistema.
Planificador de Misiones	Ofrece una interfaz para la visualización y redefinición de objetivos a través del envío de comandos de alto y bajo nivel.	Decide la siguiente acción de alto nivel para cumplir los objetivos de la misión.
Sistema Supervisor	Ofrece un visualizador de rendimiento del sistema. Permite el control de los procesos del sistema.	Realiza una monitorización de los procesos del sistema y recupera los posibles fallos que puedan ocurrir en cualquier de ellos.
Sistema de Control	Ofrece una interfaz de envío de comandos	Realiza las acciones de bajo nivel que controlan el movimiento del vehículo.
Sistema de Guiado	Selección del modo de vuelo y operación con el vehículo.	Cambia de modo de vuelo.

Tabla 3-1. Descripción de cada uno de los elementos del UAS y del HMI.

3.2.1 Monitorización de la conciencia situacional

Dentro de las funcionalidades básicas que ofrecer la interfaz para vehículos aéreos autónomos es de especial importancia la conciencia situacional denominada en inglés como *situational awareness*. Esta conciencia situacional está definida por Endsley (1988) como tres niveles incrementales de comprensión del operador del escenario [16] [18].

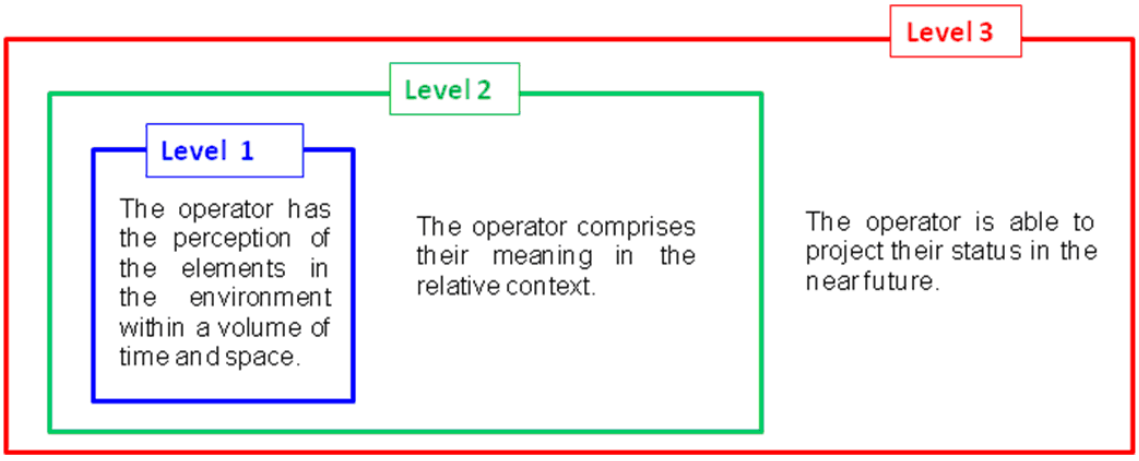


Figura 3-5 Niveles de comprensión establecidos por Endsley

Como se puede observar en la figura 3-5, el parámetro determinante en la conciencia situacional del operador es la habilidad que tiene este para comprender y proyectar un estado del entorno y poder desarrollar estados futuros que le permita tomar decisiones y actuar sobre el estado actual.

En la definición original que da Endsley se asume que el operador humano es el único agente inteligente del sistema. Considerando un UAV con mayor nivel de autonomía como es el caso, la conciencia situacional debe ser ampliada al contexto del vehículo. Un vehículo autónomo al igual que el operador debería ser capaz de tener un mínimo de conciencia situacional que le permita desarrollar nuevos objetivos y decisiones sobre un entorno y misión determinados. Por ejemplo en la re planificación autónoma se requiere detalles de información como por ejemplo el estado, la posición, o los nuevos objetivos, resultado de la re planificación.

Esta habilidad del operador depende directamente de la comunicación del operador con el vehículo autónomo a través de la interfaz, ya que en este caso es el vehículo el que ofrece esta visión de su entorno y el operador supervisa la capacidad del mismo para comprender el escenario y ser capaz de proyectar estados futuros que le permitan cumplir la misión. De esta manera, en el caso de que el vehículo aéreo no tripulado, haga una

comprensión errónea del mundo o no sea capaz de tomar de decisiones sobre una configuración del entorno inesperada, debería ser operador el que tuviera esa comprensión del entorno y ser capaz de transmitírsela al vehículo a través de la interfaz o mediante la reprogramación de algunos componentes si fuera posible.

En particular, es de especial importancia en el diseño de la interfaz proporcionar información acerca de los siguientes parámetros básicos relacionados en este aspecto. [17]:

- Posición del UAV respecto a:
  - Terreno para evitar posibles colisiones.
  - Otros vehículos en el desarrollo de misiones complejas o evitar posibles colisiones con estos.
  - Objetivos.
  - Proyecciones futuras de estas relaciones espaciales.

- **Condiciones climáticas del entorno.** Aspectos como la temperatura o viento podrían afectar al estado de la misión o la estabilidad del vehículo.

- **Estado de los componentes físicos del UAV.** Dentro de este ámbito entra en juego el correcto funcionamiento del hardware del vehículo, una correcta configuración de los dispositivos o el nivel de batería del mismo.

- **Estado del sistema a bordo del UAV.** Incluye una correcta inicialización de los procesos que conforman el software del vehículo, así como, asegurar el correcto funcionamiento de los mismos a lo largo de la misión.

- **Dinámica del UAV.** Un ejemplo podría ser la altitud, la velocidad, la orientación o el desplazamiento.

- **Progreso de la misión.** Se trata de informar acerca del cumplimiento de los objetivos de la misión.

- **Grado de confianza.** El grado de confianza del operador dependerá directamente del progreso de la misión y del nivel de autonomía, en concreto, el estado del sistema autónomo.

3.2.2 Roles del operador

Como se ha visto en capítulos anteriores, el tipo de interacción entre el usuario y el vehículo puede cambiar a lo largo del vuelo, dando lugar a distintos roles en un mismo usuario. En particular, empezando por los roles en un sistema automático podemos observar un rol mixto del usuario y el operador a lo largo del vuelo, no siendo posible un desacoplamiento casi total de la acción humana como podría ocurrir en los sistemas autónomos. En este sentido, al incrementar el nivel de autonomía del vehículo se incrementa la supervisión humana, siendo indispensable ofrecer una visión de conjunto a través de la interfaz de cada uno de los componentes que incrementan la autonomía del vehículo.

En la figura 3-6 se puede observar el tipo de interacción del usuario con el vehículo para un control manual, dónde se puede aprecia la dependencia existente de los roles de supervisión y operación para cada una de las tareas [16] [17].

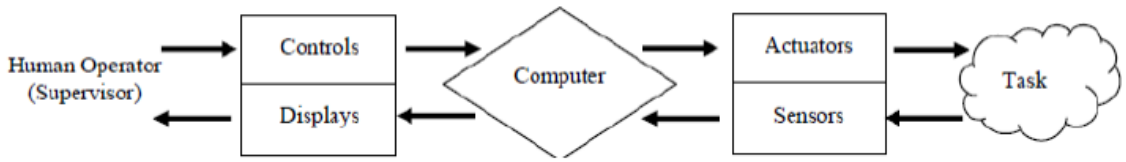


Figura 3-6. Rol del usuario y agentes principales en la interacción hombre-máquina.

Tomando como referencia la arquitectura anteriormente descrita, en un bucle interno relativo al control del movimiento del vehículo, el supervisor del sistema debe visualizar correctamente la dinámica del vehículo, comprendiendo los modos básicos de control que ofrece el sistema autopiloto del vehículo. En el segundo bucle tomando como función principal la navegación del vehículo, el usuario debe ser capaz de comprender la posición, la planificación y su ejecución a lo largo del vuelo, involucrando en el proceso, prácticamente, cualquier otro componente de la arquitectura y resultando clave para detectar fallos en componentes más internos en la arquitectura. Finalmente hay que considerar el bucle externo para la planificación del vehículo, teniendo un rol de supervisión de la misión y los objetivos que el vehículo se plantea en ésta. Paralelamente a estos tres bucles principales, existe un bucle supervisor del sistema que monitoriza el correcto funcionamiento del sistema y gestiona los fallos que pudieran darse en él [19].

En este sentido, la reducción de la implicación humana en tareas de más bajo nivel, permite involucrar al usuario en tareas con mayor demanda de atención como podría ser la supervisión de los procesos, el análisis de la situación o la toma decisiones.

3.3 Requisitos funcionales

En esta sección se describen los requisitos funcionales y no funcionales de la herramienta atendiendo a los diferentes niveles de interacción y correspondencia de funcionalidades descritos en apartados anteriores.

Para la identificación de requisitos con el usuario final del sistema se realizaron diversos prototipos exploratorios centrados en un principio en la detección y recuperación de fallos del sistema, mostrando mediante un grafo de procesos el estado de los procesos y la comunicación entre ambos. Estos resultados se validaron con los principales usuarios de la herramienta, el grupo de investigación vision4UAV de la UPM.

Este grupo de investigación se dedica principalmente a aplicar técnicas de visión por computador ofreciendo distintas aplicaciones en el área civil con UAVs como podrían ser reconstrucción 3D con imágenes, control visual, reconocimiento y seguimiento de objetos o inspección de áreas industriales.

Finalmente, se optó por un prototipo más centrado en la operación con el vehículo aéreo no tripulado, conservándose alguna de las ideas de prototipos anteriores como la supervisión del sistema y el estado del vehículo o la visualización de las entradas y salidas de los procesos. (Véase Anexo A)

La lista de requisitos funcionales que debe cumplir la interfaz, son:

Requisitos funcionales de alto nivel:

1. Ofrecer asistencia de la situación actual del sistema a bordo.
2. Operar sobre el vehículo aéreo mediante el envío de comandos.
3. Ofrecer asistencia de los objetivos de un equipo o de vehículos individuales.
4. Ofrecer asistencia sobre el estado del equipo o de vehículos individuales.
5. Controlar los niveles de autonomía de un equipo o individuales.
6. Ofrecer una interfaz que responda a los diferentes niveles de autonomía del UAV.
7. Ofrecer la posibilidad de monitorizar los procesos sistema.
8. Visualizar la información del vuelo.
9. Responder a situaciones de emergencia.

Requisitos funcionales de bajo nivel:

1. Visualización del modelo 3D del vehículo aéreo, mostrando la dinámica del mismo.
2. Visualización de un panel con el estado general del sistema.
3. Visualización gráfica de los parámetros recibidos en vuelo.
4. Recreación 3D del entorno percibido por el UAV resultado de la salida de los algoritmos de percepción.
5. Visualización del estado de los procesos del sistema.
6. Control de ejecución sobre los procesos del sistema.
7. Visualización de las imágenes recibidas por las cámaras a bordo del vehículo.
8. Visualización de los mensajes indicadores del estado recibidos del sistema.
9. Posibilidad de redefinir la misión.

La lista de requisitos no funcionales, en cambio, se ha centrado en asegurar la operatividad y usabilidad del sistema:

1. Interfaz gráfica de usuario.
2. Independiente del vehículo aéreo no tripulado.
3. Ofrece comunicación con procesos ejecutándose sobre ROS.
4. Altamente configurable.
5. Escalable.
6. Usable.
7. Eficiente en la ejecución de la misma, como en la comunicación con sistema.
8. Interoperable.

Según estos requisitos, se han identificado los siguientes componentes básicos de la interfaz de usuario, representados en la figura 3-7:

- **Interfaz de comandos.** Ayuda al usuario a operar con el UAV.

- **Editor de trayectoria.** Permite al usuario la redefinición de objetivos.

- **Monitor de estado del vehículo.** Ayuda al usuario a monitorizar el comportamiento de UAV observando la dinámica del mismo como la velocidad, posición, orientación, imágenes recibidas de la cámara o el entorno percibido.

- **Monitor de sistema**. Ayuda al usuario a monitorizar la actividad de la red de procesos, mostrando el estado de los mismos.

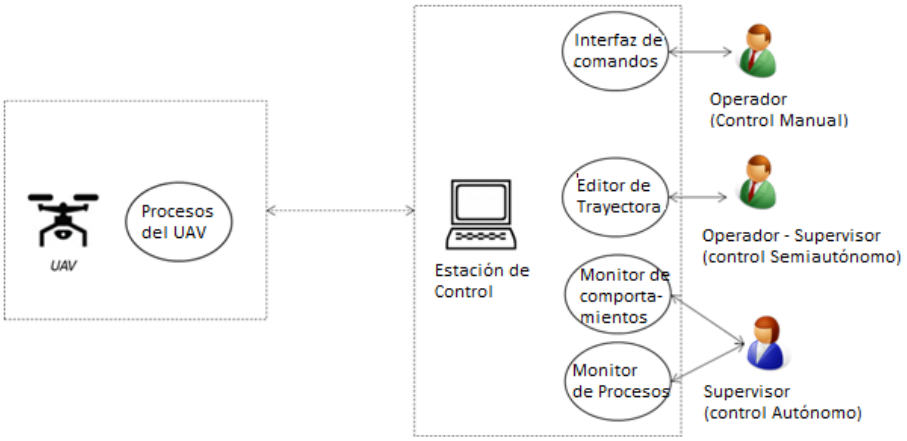


Figura. 3-7: Distribución de roles y componentes principales.

Como se puede observar en el diagrama de la figura, se han identificado 2 tipos de usuarios: el operador de vuelo y el supervisor del sistema. Estos dos roles se comunican con funcionalidades concretas que ofrece la interfaz y su grado de implicación depende del modo de vuelo y de la autonomía del sistema como se ha mencionado en capítulos anteriores.

En relación a los componentes básicos que se han descrito anteriormente, a continuación se describe una lista de componentes generales y componentes específicos de cada uno de ellos.

Los componentes principales que se han identificado en la aplicación son:

- **Panel de Control.** Presenta información del estado general del sistema como el estado de la conexión, la batería, los valores de armado y estabilizado o la acción que el vehículo está llevando a cabo en ese momento. Además permite cambiar entre los distintos modos de vuelo, incluye los comandos básicos de vuelo y permite seleccionar el vehículo en caso de que exista más de uno.

- **Visualizador de la cámara.** Ofrece imágenes de las cámaras en tiempo real.

- **Visualizador de la dinámica del vehículo.** Ofrece una representación del vehículo que ayuda al usuario a visualizar los ángulos de rotación y la altitud del mismo.

- **Interfaz de comandos.** Permite al usuario operar con el vehículo a través de básicos comandos como podrían ser *take-off* (despegar), *forward* (ir hacia adelante) o *backward* (ir hacia detrás), además de comandos extremos como podrían ser reinicializar los valores o parada de emergencia.

Monitor de Comportamiento

- **Visualizador del entorno percibido.** Ayuda al usuario a visualizar el entorno percibido por el vehículo y la trayectoria del mismo.

- **Visualizador de parámetros.** Ayuda al usuario a visualizar la información de vuelo recibida de los sensores y los parámetros estimados por los algoritmos de percepción. El usuario puede visualizar a través de curvas gráficas la evolución temporal de los mismos y comparar el valor obtenido con los distintos algoritmos.

Monitor de procesos

- **Visualizador de procesos.** Ayuda al usuario a monitorizar el estado de actual del sistema a través del árbol de procesos organizados por los subsistemas que forman la arquitectura. Además permite controlar la ejecución de los mismos con opciones como *start, stop, reset o record.*

- **Visualizador de mensajes.** Muestra el detalle de los errores que se producen en el sistema, estando por tanto ligado al subsistema supervisor del mismo.

3.4 Descripción de Componentes

En este capítulo se hace una descripción detallada de cada uno de los componentes y la interacción que tiene el usuario con cada uno de ellos.

3.4.1 Estructura general

A continuación se presenta el diseño general de la aplicación que responde a los requisitos de alto nivel propuestos en el capítulo anterior, enfocándose sobre todo en los siguientes puntos clave:

- Monitorizar el estado y el rendimiento del sistema dentro de la misión.
- Actuar en caso de fallo para redefinir o abortar la misión.
- Visualización de la información de vuelo recibida de cada uno de los componentes de la arquitectura.

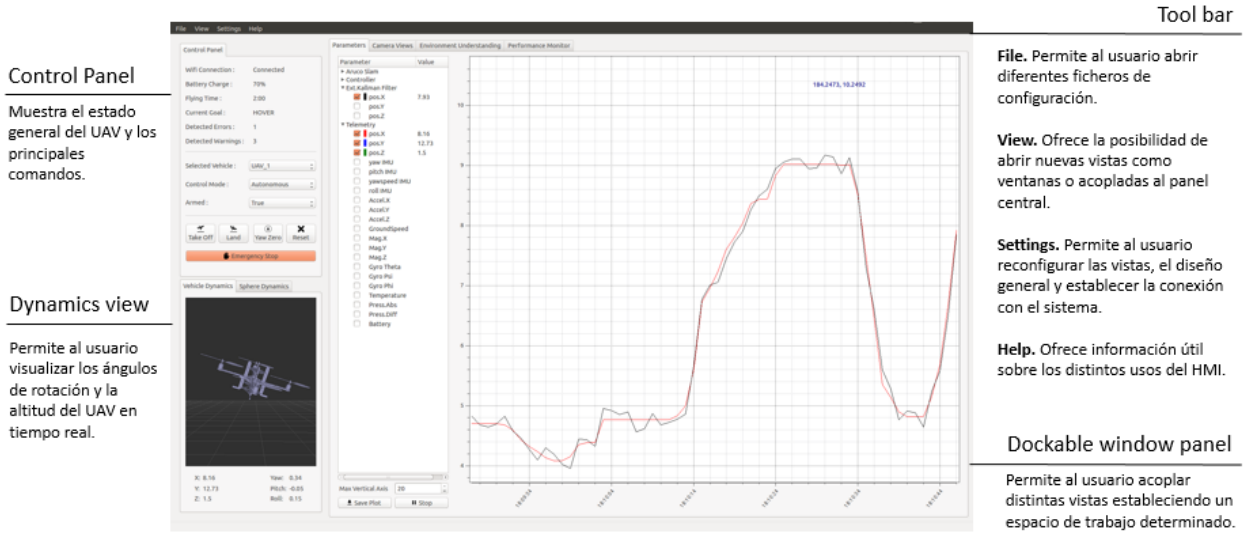


Figura 3-8 Estructura general

Como se puede observar en la figura 3-8 la herramienta tiene 3 zonas principales de interacción: el panel de control (*Control Panel*), la visualización de la dinámica (*Dynamics View*) y el panel central (*Dockable window panel*). En el panel central se puede navegar entre distintas vistas en las que se incluyen la vista de parámetros (*Parameter View*), la vista de las cámaras (*Camera View*), la visualización del entorno percibido (*Environment Understanding*) y el monitor de rendimiento del sistema (*Performance Monitor*).

3.4.2 Panel de control

El panel de control representa uno de los componentes principales de la aplicación, a través de él se informa al usuario del estado general del sistema, proporcionando la información que se considera crítica y que requiere mayor demanda de atención por parte del usuario.

Este panel de control está divido en dos zonas; una dedicada a la interacción y otra zona dedicada a informar al usuario del estado general del sistema.

En la zona de interacción podemos encontrar comandos básicos de alto nivel como podría ser la selección del vehículo o el cambio de modo de operación con el mismo y comandos de bajo nivel estrechamente relacionados con el controlador del vehículo como el aterrizaje o despegue del vehículo.

En la zona de información se proporciona retroalimentación de parámetros básicos como podría ser el estado de la conexión (*Wi-Fi Connection*) inicialmente desconectado, tomando el valor conectado cuando el usuario se conecta al sistema, el indicador de la batería (*battery charge*) o el tiempo que lleva volando el vehículo (*Flight Time*) tomando como valor cero el momento en el vehículo despega y considerando la necesidad de realizar un despegue manual del vehículo a través de la aplicación.

Para que el usuario pueda interaccionar con este panel previamente debe haber armado el vehículo a través del valor armado (*armed*) con el botón de selección que el panel ofrece. Este botón actúa de seguro para evitar, por ejemplo, que despegue el vehículo en el caso de que no todos los procesos se hayan inicializado correctamente.

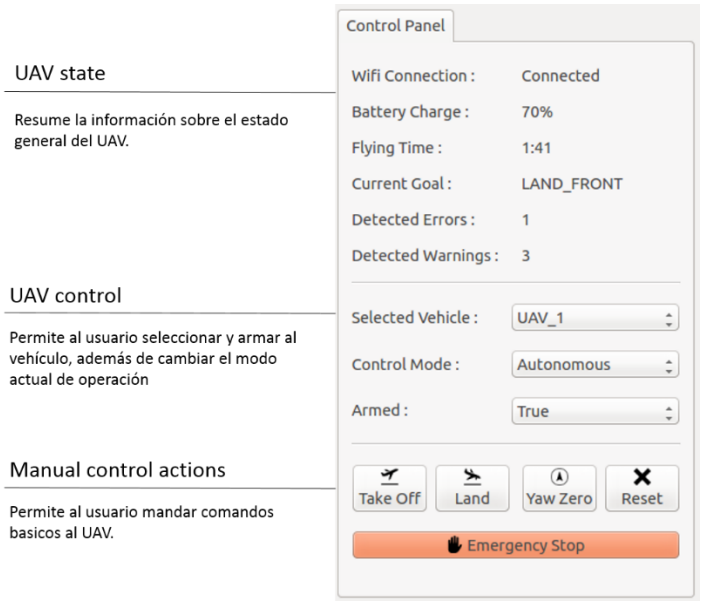


Figura 3-9. Panel de control.

El panel de control, además ofrece a través de un botón de selección los siguientes modos de operación relacionados con la autonomía del UAV:

Manual. El usuario opera y supervisa el UAV para completar la misión enviando comandos básicos como podrían ser *take-off, forward, backward,* etc al UAV.

Guiado. El usuario guía al UAV para completar la misión definiendo objetivos a través de *waypoints.*

Autónomo. El usuario supervisa la acción actual y el estado del sistema.

3.4.3 Visualizador de la dinámica del vehículo

El visualizador de la dinámica del vehículo está formado por una representación 3D del vehículo, eligiéndose como modelo representativo el ofrecido por *AscTec Pelican* y una esfera con 3 ejes fijos y otros 3 ejes rotativos. Los 3 ejes fijos o ejes locales representan el sistema de coordenadas de referencia, mientras que los ejes rotativos o ejes globales representan los cambios de orientación respecto a este sistema de coordenadas.

En ambas representaciones también se ha proporcionado la visualización del plano, utilizándolo como horizonte para visualizar los movimientos de traslación en el eje y, lo que correspondería a la altitud del vehículo.

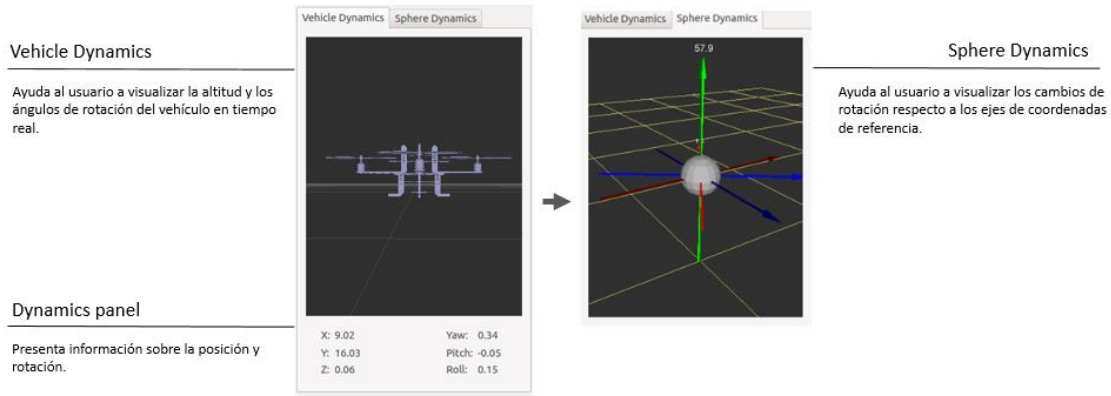


Figura 3-10. Visualizador de la dinámica del vehículo.

Este tipo de movimientos de rotación y traslación son dependientes de las velocidades de giro de cada uno de los motores, funcionando a su vez como indicadores del estado actual de estos y permitiendo al usuario realizar el siguiente cambio de movimiento.

Los movimientos de rotación sobre los 3 ejes de coordenadas que se utilizan para controlar el cuadricóptero son:

- **Roll:** movimiento de rotación sobre el eje X. Su variación permite desplazar el cuadricóptero hacia la derecha o la izquierda.
- **Pitch:** movimiento de rotación sobre el eje Y. Su variación permite desplazar el cuadricóptero hacia delante o detrás.
- **Yaw**: movimiento de rotación sobre el eje Z. Su variación permite rotar el cuadricóptero hacia la derecha o la izquierda.

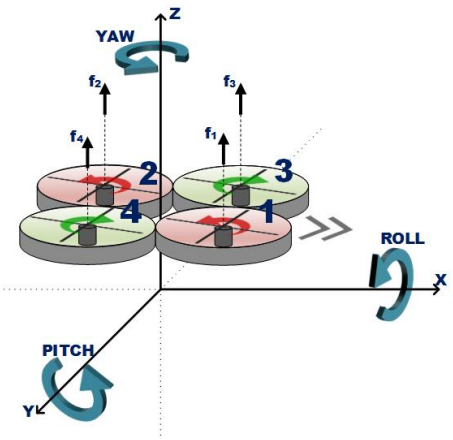


Figura 3-11. Movimientos de rotación.

Estos tres movimientos de rotación se expresan como ángulos, generalmente en radianes.

Además de las dos representaciones 3D, se incluye un panel debajo de cada una de las representaciones donde se indica el valor númerico con dos decimales de estos movimientos.

3.4.4 Visualizador del entorno percibido

El visualizador del entorno percibido realiza una abstracción de la escena sobre la que se desarrolla la misión 7 de la competición del IARC descrita en el capítulo 5 de este documento, siendo el único componente diseñado específicamente para ésta.

Teniendo en cuenta las reglas de esta competición. La información que se representa en esta representación es la siguiente:

- Posición estimada del vehículo dentro de una rejilla representativa y que no forma parte del escenario real para medir dicha posición.

- Robots detectados dentro del área de visión del vehículo.
- Intersecciones detectadas del plano real sobre el que desarrolla la misión.

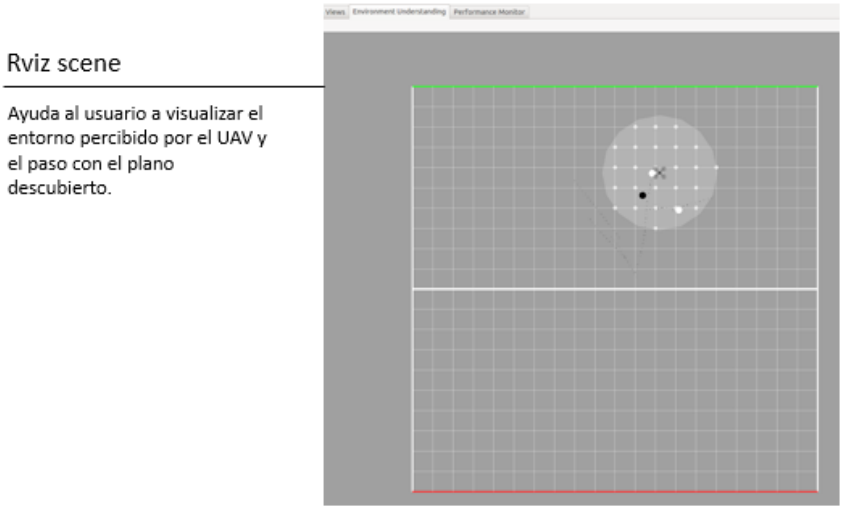


Figura 3-12. Entorno percibido

3.4.5 Monitor del rendimiento del sistema

El monitor del rendimiento del sistema está formado por dos componentes básicos: el visualizador de procesos y el visualizador de mensajes.

La función del visualizador de procesos es informar al usuario del estado general de los procesos del sistema, pudiéndose encontrar los siguientes estados:

- Inicializando (*Initializing*). El proceso se está preparando para ejecutarse.
- Corriendo (*Running*). El proceso se está ejecutando normalmente en el sistema.
- Esperando (*Waiting*). El proceso entrará en este estado cuando se requiera información crítica para continuar su ejecución normal en el sistema.
- Durmiendo (*Sleeping*). El proceso no ejecuta ninguna instrucción pero está preparado para volver en cualquier momento a su ejecución normal.
- Parando (*Stopping*). La ejecución normal del proceso ha sido interrumpida.
- Recuperando (*Recovering*). El proceso se está recuperando de un error inesperado en el sistema

Este estado va a acompañado por el tiempo de inicio en el sistema tomando como referencia el tiempo de vuelo que indica el panel de control y el tiempo de fin de ejecución en el sistema.

Además de esta información ofrece información acerca del rendimiento en general como podría ser la carga de procesamiento medida como en porcentaje de CPU que ocupa y el número de hilos de ejecución o la memoria que consume dicho proceso.

El visualizador de mensajes ofrece información acerca del estado de los procesos organizada por tiempo a través de un identificador temporal o *timestamp*, está información se recibe del proceso supervisor del sistema y está clasificada en los siguientes niveles de severidad:

`DEBUG`

- Esta información se envía en estado de validación del sistema y sirve para realizar pruebas unitarias que verifiquen la robustez de cada uno de los procesos del sistema.
- Ejemplos:
    - Recibido mensaje en el *topic* X desde el proceso Y.
    - Se han enviado 20 bytes a través del socket 10.

`INFO`

- Información emitida por el propio programador en el código del sistema y que puede ser útil al usuario de la aplicación,
- Ejemplos:
    - “Nodo inicializado”
    - "Se ha publicado en el topic X con el mensaje Y”.
    - "Nuevo subscriptor Y en el *topic* X”.

`WARN`

- Información que el usuario pueda encontrar alarmante y que pueda afectar al correcto funcionamiento del sistema.
- Ejemplo:
    - "No se pudo cargar la configuración por defecto del fichero en <path>. "

`ERROR`

- Un error serio pero recuperable ha ocurrido.
- Ejemplos:
    - “No se ha recibido actualización del *topic* X en 10 segundos. Parando el proceso hasta que se consiga transmisión”"
    - "Se ha recibido un valor inesperado en la transformación de movimiento X.. El valor de esta transfromación ha sido ignorado.

`FATAL`

- Algo irrecuperable ha pasado.

- Ejemplos:
    - ¡Los motores del vehículo han sobrepasado el umbral de temperatura!
    - El ala X del vehículo ha dejado de funcionar correctamente.

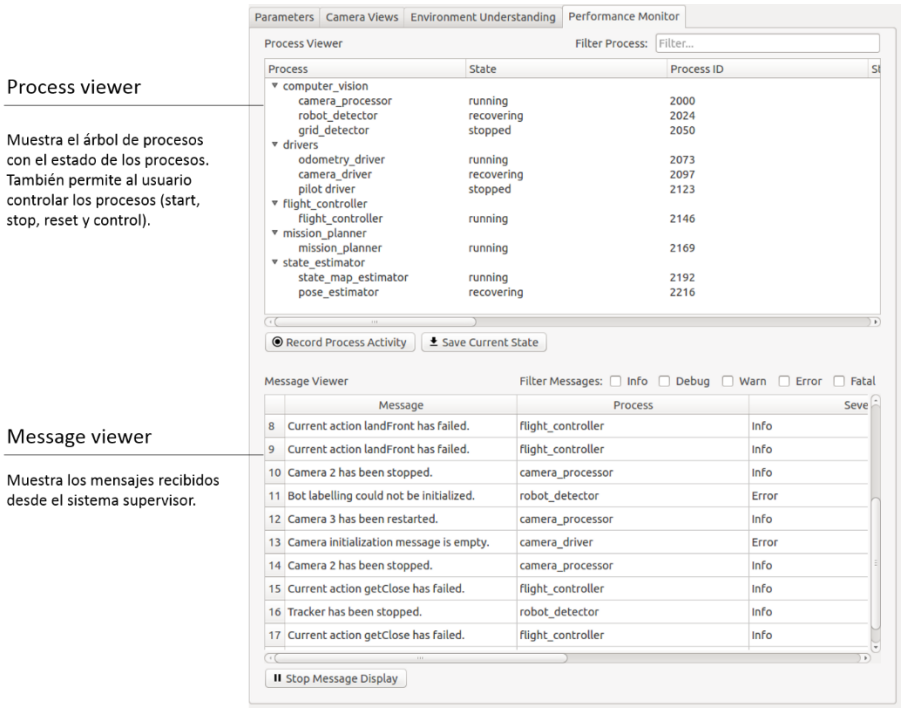


Figura 3-13. Monitor del rendimiento del sistema

3.4.6 Visualizador de parámetros

El sistema encargado de la visualización de parámetros está formado por un navegador de *topics* o canales de comunicación en ROS a través del cual se muestran los distintos parámetros que se envían a través de esos *topics* definidos y encapsulados mediante mensajes.

Cada uno de estos parámetros despliega una curva gráfica identificada por colores fácilmente diferenciables que se van asignado dinámicamente a petición del usuario, evitando el problema de llegar a tener colores similares en la gráfica con pocas curvas. Mientras que al lado del nombre de cada parámetro se muestra el valor numérico que va tomando el parámetro a lo largo del tiempo, llegando a representar hasta 4 decimales.

El sistema de visualización de curvas gráficas ofrece la posibilidad de observar la evolución temporal dentro de un intervalo de tiempo determinado, tomando como valor

inicial la hora actual del sistema, desplazándose a medida que pasa el tiempo de derecha a izquierda. El eje de las Y, en cambio, se ha diseñado de manera configurable debido a la diversidad de parámetros y rango de valores que se puede obtener de los mismos, este valor se puede configurar mediante un selector o haciendo zoom directamente en el *canvas*. Además de la posibilidad de hacer zoom sobre la gráfica el usuario también tiene la posibilidad de parar la evolución temporal de las curvas gráficas que se despliegan en él y/u obtener el valor de un parámetro determinado en un instante de tiempo determinado llevando el cursor a la posición deseada en los ejes.

Este tipo de visualización es especialmente útil y ayuda al usuario a observar el error cometido entre diferentes algoritmos de percepción al estimar la posición de vehículo aéreo no tripulado, como para verificar el correcto funcionamiento de los sensores o visualizar los detalles de la trayectoria del vehículo en un intervalo de tiempo determinado.

Como características adicionales, se ha añadido la posibilidad de guardar el estado del sistema de visualización de curvas gráficas en distintos formatos como *pdf o PostScript.*

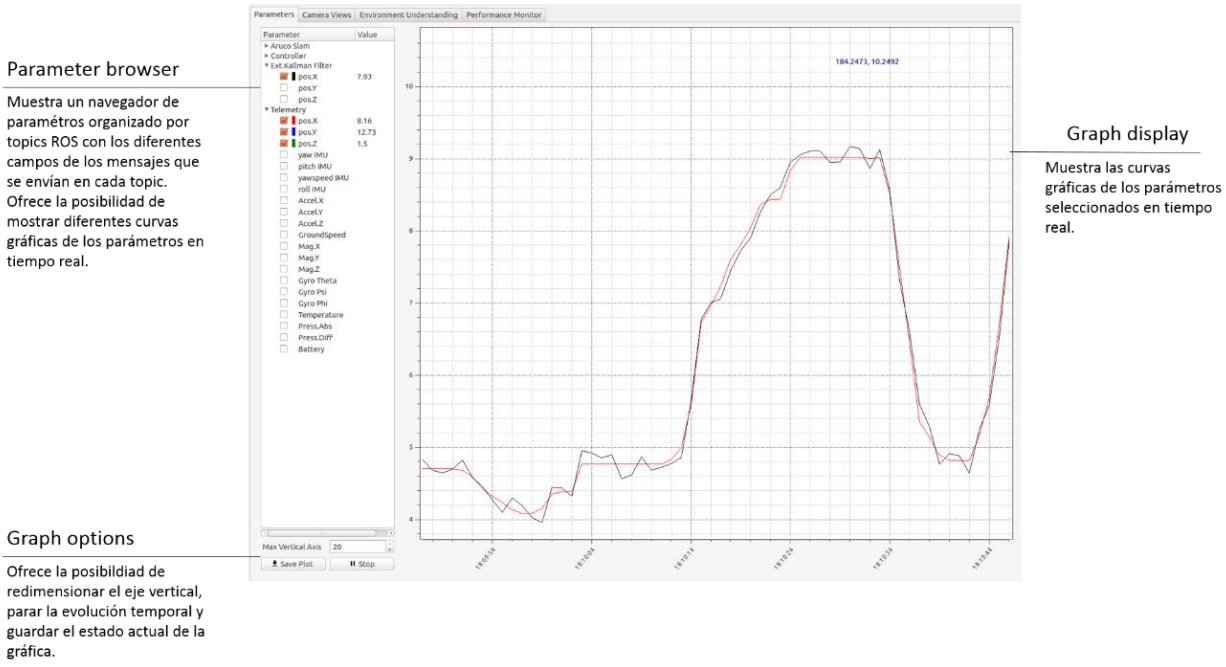


Figura 3-14. Visualización de la trayectoria del vehículo.

3.4.7 Visualizador de la cámara

La función de este componente es ofrecer imágenes y/o vídeo capturados por el vehículo aéreo durante el vuelo. Para la visualización de las distintas cámaras se ofrecen diferentes opciones de visualización.

En la figura 3-15 se puede observar la opción de visualización principal de este componente con una cámara seleccionada como principal visualizando imágenes de esta con mayor tamaño, teniendo las imágenes del resto de cámaras con menor tamaño y la posibilidad de intercambiar la vista con cualquiera de estas.

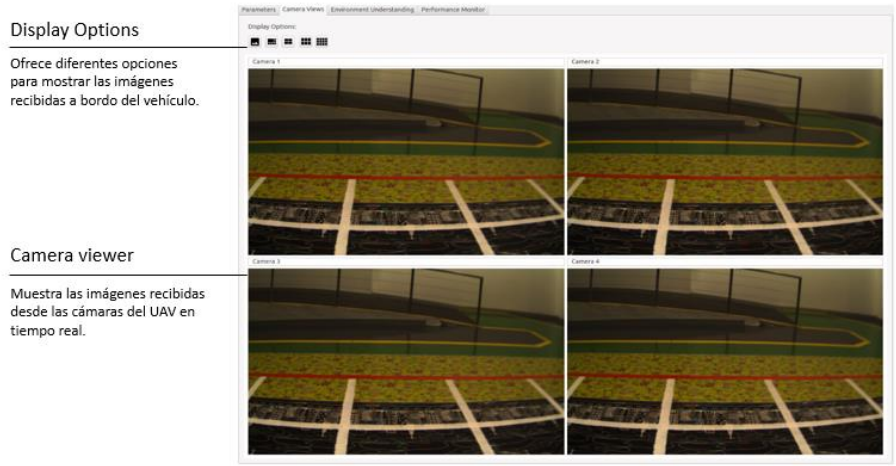


Figura 3-15. Visualizador de cámaras

3.4.8 Consola de comunicación

Este componente ofrece información acerca del estado de la aplicación en particular. Mostrando la recepción de mensajes junto con un identificador temporal dependiente de ROS (*Robot Operating System*) descrito en el siguiente capítulo. Además de esto guarda un log con los errores producidos en la aplicación resultado de una mala comunicación de esta con el sistema a bordo del vehículo.

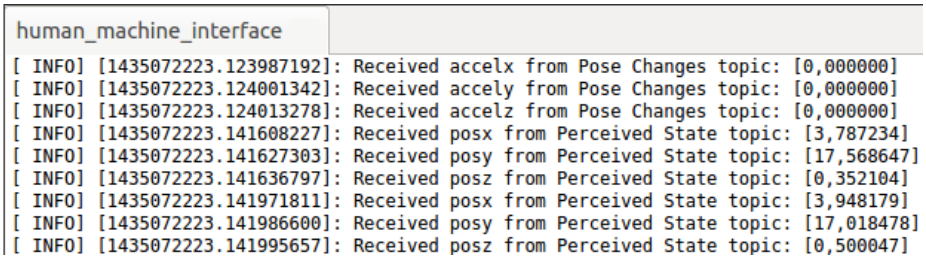


Figura 3-16. Consola de comunicación

3.4.9 Interfaz de comandos

Este es uno de los componentes principales de la aplicación, se trata de una interfaz a través del teclado con la que el usuario interacciona con el vehículo aéreo mediante el envío de comandos básicos descritos en la siguiente tabla :

Tecla	Comando de control
'h'	Hover (planear)
'l'	Land (aterrizar)
'return'	Parada de emergencia
'\''	Poner yaw a 0.
'r'	Poner los comandos a cero
'tab'	Cambiar modo de control
'a'	Yaw en contra de las agujas del reloj.
'd'	Yaw sentido las agujas del reloj.
'w'	Pitch sentido las agujas del reloj.
's'	Pitch en contra las agujas del reloj.
'q'	Roll en contra las agujas del reloj.
'e'	Roll en sentido las agujas del reloj.
↑ (tecla de navegación)	Moverse hacia adelante.
↓ (tecla de navegación)	Moverse hacia detrás del vehículo. (retroceder)
→ (teclado numérico)	Moverse hacia la derecha en el eje y.
← (teclado numérico)	Moverse hacia la izquierda
↓ (teclado numérico)	Moverse hacia detrás.
↑ (teclado numérico)	Moverse recto.

Figura 3-2. Tabla de comandos

3.4.10 Conexión

Para la inicialización de los componentes de la interfaz el usuario previamente se debe conectar al sistema a bordo del vehículo y configurar la conexión del sistema a través del panel de conexión que ofrece la interfaz.

La ventana de comunicación permite al usuario conectarse a diferentes máquinas sobre las que se esté ejecutando ROS (*Robotic Operating System*). ROS está diseñado para actuar como un sistema distribuido. Esto permite conectar diferentes máquinas como el HMI en la estación de control terrestre con otros nodos o procesos a bordo del UAV.

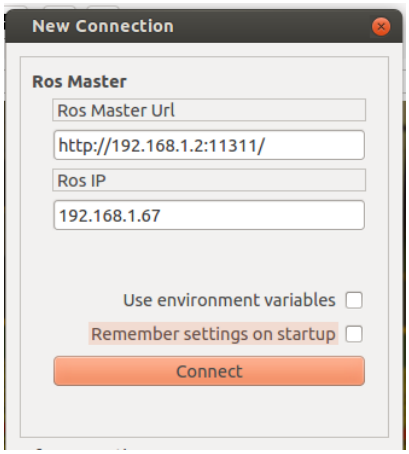


Figura 3-17 Ventana de comunicación

De esta manera en el diseño del HMI se ha proporcionado una ventana de configuración de conexión con los siguientes campos:

- **ROS_MASTER_URI.** Es una variable de entorno del sistema que guarda la url del nodo master. Esta variable de entorno le dice a los nodos donde está alojado el proceso máster encargado de la comunicación de los mismos.
- **HOST_IP.** Es la dirección IP de la máquina donde está ejecutándose el HMI.

La opción "*use environment variables*", en cambio, permite la conexión automática del HMI siempre y cuando este sea lanzado desde la terminal del computador.

3.4.11 Estado general de la aplicación

Además de los distintos componentes gráficos se debe informar al usuario en todo momento del correcto funcionamiento de la aplicación. En el diseño de la interfaz se ha intentado minimizar estos errores proporcionando mecanismos de seguridad como podrían ser el valor de armado o el botón de conexión evitando una conexión automática de la aplicación a cualquiera de los sistemas presentes en ROS o el despegue del vehículo cuando los procesos no han sido aún inicializados correctamente.

4 IMPLEMENTACIÓN

A continuación se hace una descripción del diseño e implementación de la interfaz de usuario a nivel general describiendo las técnicas y herramientas utilizadas en el diseño y proceso de construcción de la arquitectura software.

4.1 Arquitectura Software

La siguiente arquitectura ofrece una implementación basada en 3 niveles o capas; la capa de presentación, la capa lógica y la capa de comunicación.

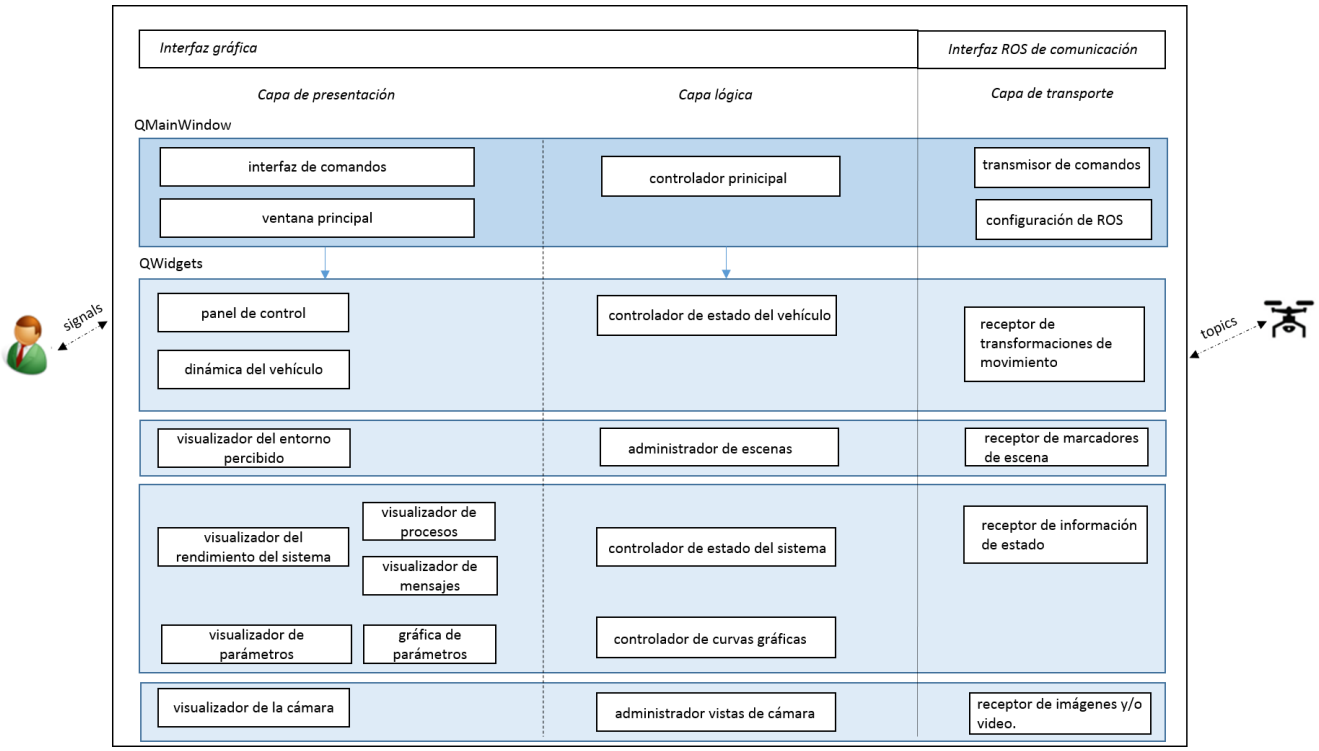


Figura 4-1. Arquitectura software

En la capa de presentación se implementan los diferentes elementos gráficos que ofrece la aplicación implementados mediante una jerarquía de elementos que heredan del componente principal *QMainWindow*.

La capa lógica, en cambio, gestiona la modificación dinámica de vistas y acceso a memoria mediante la recepción, captura y envío de señales de sistema.

Por último, la capa de comunicación es la que ejecuta sobre ROS, para comunicarse con el vehículo aéreo no tripulado a través de la recepción y envío de mensajes que ofrece este sistema.

El diseño de la arquitectura software está basado en un patrón MVC (*Model-View-Controller*). Este patrón de diseño define controladores que se encargan de gestionar la lógica de cada una de las vistas de la interfaz, implementando las funciones que se ejecutan como resultado de la recepción de un evento producido por la interacción del usuario con la interfaz.

Como ventajas de este patrón de diseño se destaca una mejora en la experiencia de usuario a través de la modificación de dinámica de componentes y una mejor gestión de acceso a memoria de los distintos hilos de ejecución.

El uso de este patrón de diseño ha sido de especial importancia debido al carácter asíncrono que se encuentra tanto en el nivel de comunicación mediante la ejecución de las retro llamadas a funciones en ROS ante la recepción de mensajes como a nivel de interfaz resultado de la interacción del usuario con la misma.

La sincronización de lecturas de la aplicación y escrituras de los hilos de ejecución sobre ROS se ha realizado manteniendo una sincronización interna mediante el envío de señales. De esta manera ante la recepción de un mensaje por un canal de comunicación determinado se envía una señal al controlador o proceso encargado de la lectura de ese espacio de memoria que gestiona el cambio las vistas asociadas a ese controlador.

En la capa de comunicación también se ha establecido una sincronización interna para evitar reservar espacio de memoria de forma ineficiente, implementando una subscripción dinámica a *topics* en ROS mediante el envío de señales que activan dicha subscripción ante la petición del usuario de visualización de una determinada información. Sin embargo, una vez establecida dicha comunicación la recepción de información se hace de manera asíncrona hasta que se captura una señal resultado de una petición del usuario de dejar de visualizar dicha información.

4.2 Estructura de clases

El diseño utilizado para la estructuración de clases de interfaz de usuario o vista de la aplicación se ha utilizado el patrón de diseño *composite*. Este patrón de diseño es ampliamente utilizado en la implementación de interfaces de usuario, permitiendo crear una taxonomía de elementos que pueden ser tanto elementos simples como un conjunto formado por estos elementos. Esta manera de agrupar y clasificar las clases proporciona mucha flexibilidad software, ya que permite un número muy alto de configuraciones posibles.

El uso de este patrón junto con la herencia y el paradigma orientado a objetos, ha hecho posible la creación, modificación y eliminación de elementos de manera aislada, sin afectar al resto de clases ni funcionalidades de estas, ofreciendo así un desarrollo flexible y altamente configurable adaptado a los posibles cambios del cliente o usuario de la aplicación. Además de esto, este patrón de diseño proporciona la capacidad al sistema de definir restricciones sobre las configuraciones que se pueden adoptar o las que se encuentran prohibidas por el propio modelo de datos.

A continuación se muestra en la figura 4-2 un diagrama que ejemplifica el uso de este patrón de diseño en la arquitectura software. Este diagrama es una simplificación del diagrama UML que se proporciona en el anexo C de este documento.

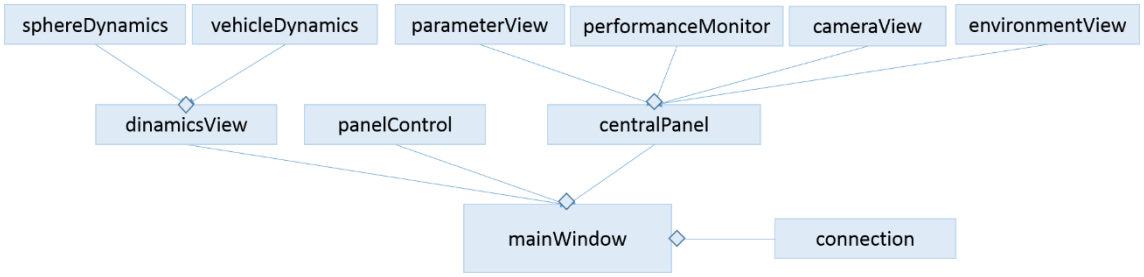


Figura 4-2. Estructura jerárquica de las clases principales de la herramienta

Las distintas perspectivas y componentes principales de la interfaz se implementan a través de lo que se conoce como *widgets*, siendo la clase principal de la cual heredan los objetos específicos que componen la interfaz como podría ser objetos del tipo *frames, tab widgets, bar widgets* etc en la interfaz de usuario.

La distribución de los componentes en la interfaz se hace a través de la composición de clases específicas mediante la creación de objetos de tipo *layout*, ofreciendo diferentes niveles de distribución y una jerarquía de elementos organizados en una malla que mantiene la simetría entre los diferentes componentes de la interfaz.

4.3 Herramientas de desarrollo

Para la implementación de la interfaz se ha utilizado ROS como sistema soporte y de comunicación con el sistema a bordo del vehículo.

Los componentes gráficos de la interfaz, en cambio, se han implementado independientes al entorno de ROS, utilizando la biblioteca multiplataforma Qt y el lenguaje de programación C++.

4.3.1 ROS – *Robot Operating System*

ROS (*Robotic Operating System*) es un middleware que introduce un conjunto de herramientas y librerías para ayudar al desarrollo del software de aplicaciones en el campo de la robótica. Permite una abstracción del hardware y los controladores del dispositivo, proporcionando bibliotecas que facilitan la comunicación a través del paso de mensajes entre los diferentes procesos.

Originalmente se desarrolló con el nombre de *switchyard* por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto Robot con Inteligencia Artificial de Stanford. Actualmente el proyecto lo mantiene Willow Garage bajo licencia BSD [20].

Este sistema operativo robótico se basa en una arquitectura de grafos sobre el procesamiento de nodos que se comunican a través de *topics*, siguiendo un paradigma de comunicación editor-subscriptor. Estos *topics* o canales de comunicación transmiten mensajes que son definidos según una especificación establecida por ROS y que sirven como interfaz de comunicación entre uno o varios nodos.

Para la gestión de comunicación entre nodos, existe un nodo máster denominado *roscore* que implementa un servicio XMLRPC que envía al nodo o proceso que se subscribe la dirección ip y puerto donde se está ejecutando el nodo que publica por ese canal de comunicación determinado. De esta manera el nodo o proceso que desea publicar un mensaje tiene que enviar previamente su dirección *ip* y puerto al *roscore* para que pueda identificarlo en el sistema.

El tipo de comunicación que ofrece ROS es de naturaleza asíncrona y anónima ejecutándose retrollamadas que se encargan de procesar los mensajes publicados en un

topic determinado, sin embargo, ROS ofrece también la posibilidad de establecer una comunicación síncrona y bloqueante a través de lo que se conoce como servicios ROS.

Esta comunicación está accesible a través de una librería cliente que se denomina API-Slave implementada en los lenguajes C++ (*Roscpp*) Python (*Rospy*) que permite la implementación de sistemas robóticos denominados *stacks*. Los *stacks* públicos forman la suite de paquetes de ROS que implementan funcionalidades como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

4.3.2 Biblioteca multiplataforma Qt

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. Fue desarrollada por Nokia cómo software libre y de código abierto a través del proyecto *QProject* [20].

Qt también es utilizada en KDE, entornos de escritorio para sistemas como GNU/Linux o FreeBSD, sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.

La biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos y soporte de red. Además de esto ofrece una API multiplataforma unificada para la manipulación de archivos y para el manejo de ficheros, además de estructuras de datos tradicionales.

Por otra parte, existen *bindings* de Qt que permite utilizar la librería con otros lenguajes de programación como podrían ser Java (*Jambi*), Python *(PyQt*), PHP (*PHP-Qt*) o C# (*Qyoto*) entre muchos otros.

Está biblioteca se puede combinar con otras librerías más específicas como la biblioteca *Qwt* o *Qt Widgets for Technical Applications* utilizada para la visualización de la gráfica de parámetros en 2d. Esta biblioteca es ampliamente utilizada para aplicaciones con perfil técnico, proporcionando deslizadores, barras, brújulas, diales, termómetros, ruedas para controlar valores de visualización o visualización de gráficas.

4.4 Implementación de los Componentes

En el siguiente capítulo se hace una descripción del proceso de implementación llevado a cabo para cada uno de los componentes que forma la interfaz.

4.4.1 Visualizador de parámetros

En la implementación del visualizador de curvas gráficas se ha utilizado la librería Qwt o *Qt Widgets for Technical Applications* que forma parte de la suite de Qt [21].

Esta librería permite la visualización de gráficos 2D con un conjunto de funciones que modifican el *canvas* y muestran curvas gráficas. La implementación de curvas gráficas se ha realizado mediante una *shallow copy* del espacio asignado al parámetro que representa la curva en cuestión. Para ello se ha utilizado un puntero de desplazamiento para desplazar las posiciones en el buffer de llenado ante la recepción de un nuevo valor obteniendo una curva gráfica con una evolución temporal.

En la actualidad los canales de comunicación de la arquitectura actual a través de los que se recibe la información no tienen una frecuencia fija, sufriendo pequeñas variaciones en el tiempo y pudiendo variar significativamente dependiendo del tipo de mensaje definido por el programador del sistema. Aunque es posible establecer una frecuencia determinada a la que debe publicar un topic mediante la implementación de colas de prioridad que realiza internamente el sistema, en la práctica no suele resultar eficiente, optando por una solución basada en la identificación de mensajes mediante *timeStamps.*

Para resolver este problema, teniendo en cuenta el estado inicial de pruebas en el que se encuentra la aplicación donde el entorno de pruebas utilizado no aporta mucha complejidad en este sentido, se ha optado por establecer una frecuencia de muestreo inferior a la que publican los topics, a través de la implementación de un temporizador interno.

Sin embargo, para la visualización de los valores en el navegador de parámetros se ha optado por utilizar la máxima de las frecuencias de la frecuencia de los parámetros que se están visualizando en un momento determinado, enviando señales continuas ante la recepción de mensajes.

4.4.2 Visualizador de la dinámica del vehículo

Para la implementación de este componente se ha utilizado *Rviz* como sistema de visualización y la librería *Tf* [19] [21] para la publicación de transformaciones que son procesadas por este sistema de visualización.

La librería *Tf* forma uno de los paquetes principales del entorno de *ROS* permitiendo el seguimiento de las diferentes estructuras de coordinación (*coordinate frames*) en las que se componen un modelo de robot descrito mediante el formato URDF. Esta librería mantiene la relación entre las distintas estructuras y permite mandar a estas estructuras transformaciones de movimientos de rotación o translación en el plano.

El modelo utilizado para la visualización 3D del vehículo ha sido el modelo de quadrotor *AscTec Pelican* proporcionado por la empresa *Ascending Technologies* en el formato Collada mantenido por el grupo *Khronos Group* que se importa en *Rviz* a través de un fichero *xml.* Para la implementación de la esfera se ha diseñado un *xml* específico para esta herramienta.

4.4.3 Visualizador del entorno percibido

Para la visualización del entorno se ha utilizado también la herramienta de visualización 3D, Rviz. Esta herramienta ofrece distintos sistemas de visualización, permitiendo al programador introducir objetos 3D mediante la definición de figuras geométricas a través de lo que se conoce como *markers* y la definición de *topics* a través de los cuales se publican los cambios de posición que resultan en el movimiento de las distintas figuras geométricas.

Sin embargo, *Rviz* no sólo es una herramienta de simulación, si no que funciona como una potente herramienta de visualización del entorno percibido mediante los sensores y los algoritmos de percepción del robot, a través de un sistema de configuración de tipos de visualización. Dentro de las posibilidades de visualización que ofrece Rviz podemos encontrar los siguientes tipos [19]:

- *Axes.* Muestra un conjunto de ejes con coordenadas (x,y,z) en colores verde, rojo y azul.
- *Effort.* Muestra la fuerza aplicada en cada una de las articulaciones del robot como círculos con flechas alrededor de las articulaciones.
- *Camera.* Crea un nuevo panel de renderizado superpuesto para la visualización de las cámaras del robot.
- *Grid.* Muestra una rejilla centrada en el origen.

- *GridCell*. Muestra una rejilla con los obstáculos que se encuentran dentro del mapa de corto alcance percibido por el robot.
- *OccupancyGrid*. Muestra el mapa del entorno con las configuraciones de ocupación de los obstáculos mediante la asignación de probabilidades de ocupación a cada celda del mapa sobre el conjunto de medidas y estimaciones de posición del robot tomadas en un espacio de tiempo determinado.
- *Image*. Crea una nueva ventana para la visualización de las imágenes recibidas por la cámara.
- *Laser Scan.* Muestra los datos recibidos de un scaner láser, con diferentes opciones de renderizado y acumulación de datos.
- *Map*. Muestra un mapa estático sobre el plano de tierra.
- *Markers*. Permite mostrar formas primitivas como figuras geométricas a través de topics.
- *InteractiveMarkers.* Permite mostrar objetos 3D desde uno o más servidores interactivos, permitiendo al usuario interaccionar con ellos a través del mouse.
- *Path*. Muestra la ruta o paso que ha seguido un robot.
- *Point*. Dibuja puntos como pequeñas esferas.
- *Pose*. Muestra la posición como una flecha o través de ejes.
- *Pose Array*. Muestra una nube de flechas para cada uno de los arrays de posiciones.
- *Point Cloud*. Muestra los datos recibidos de la nube de puntos con diferentes opciones de renderizado o acumulación.
- *Polygon*. Dibuja la traza de un polígono.
- *Odometry*. Acumula las posiciones estimadas por los algoritmos de odometría en el tiempo.
- *Range*. Muestra conos que representan la información recibida del sonar o de los sensores IR de rango.
- *RobotModel*. Muestra una representación visual del modelo 3D cargado.
- *Tf.* Muestra la jerarquía de transformaciones.

Para la integración de esta herramienta con la interfaz se ha utilizado la librería pública que ofrece ROS para acceder a las distintas funcionales que ofrece *Rviz*, denominada *librviz*, llegando a poder reconfigurar por completo el entorno utilizando las funcionalidades que está ofrece.

Esta herramienta al igual que el resto de componentes gráficos que ofrece ROS se programan en forma de *plugins* creados por la comunidad de usuarios para visualizar e interaccionar a través de una interfaz gráfica con las principales características que ofrece ROS a través de la línea de comandos. Sin embargo, el nivel de complejidad que introducen estas herramientas y dependencia con el sistema ROS, obliga al usuario a

conocer muy bien el entorno, estando destinadas la gran mayoría a depuración de procesos en ROS y realización de pruebas unitarias.

4.4.4 Monitor del rendimiento del sistema

Este componente está ligado al tipo de arquitectura del sistema, diseñándose para representar la información recibida del subsistema supervisor.

La implementación del visualizador se basa en dos tablas implementadas como *QTableWidgets* modificándose dinámicamente el estado ante la recepción de mensajes por parte del subsistema supervisor.

Para la configuración inicial del árbol de procesos de forma dinámica, previamente se pide la lista de procesos (activos y no activos) al sistema supervisor, para posteriormente ir actualizando la información con los mensajes que este le envía.

Además de la visualización de estados, la interfaz también ofrece control sobre la ejecución de los procesos, permitiendo al usuario inicializar, reiniciar y parar procesos a través de llamadas a procesos ROS que implementan esos servicios. Para ello se ha diseñado un cliente ROS que llama a estos servicios bajo petición del usuario.

A diferencia, de los nodos ROS convencionales, estos servicios implementados por un único nodo ROS ofrecen la posibilidad de establecer una comunicación bloqueante del tipo pregunta/ respuesta a través de lo que se conoce como *ROS services*, Estos servicios además ofrecen retroalimentación periódica de forma continua a través de la librería de ROS *actionlib a* través de la cual se envía el estado del proceso que lo implementa.

4.4.5 Visualizador de la cámara

Este componente se ha diseñado para visualizar las imágenes recibidas del sistema a bordo del vehículo. En la implementación del mismo se ha seleccionado las librerías de *cv_bridge* y *image_transport* de ROS.

- *cv_bridge.* Es una biblioteca que convierte los mensajes de imágenes ROS en imágenes *OpenCV* y viceversa.
- *image_transport.* Permite la publicación y subscripción de imágenes. Soporta la conversión entre diferentes formatos y la compresión de imágenes a baja resolución.

En la recepción de imágenes se transforma el formato de los mensajes de entrada de ROS en el tipo *QImage* ofrecido por la librería *Qt* para la visualización de las imágenes en la interfaz.

4.4.6 Interfaz de comandos

Para la implementación de la interfaz de comandos se ha utilizado la propia librería *QtKeyEvent* que pertenece a la suite de librerías que ofrece la librería *Qt* descrita al inicio de este capítulo.

5 VALIDACIÓN

En este capítulo se describe el proceso llevado a cabo para verificar que la HMI cumple con los objetivos propuestos. Este proceso está compuesto de diferentes verificaciones que prueban la validez de la herramienta.

El proceso de validación del HMI es de especial importancia ya que este sistema está diseñado para interactuar directamente con los controladores del UAV a través de ROS, de esta manera cualquier fallo en el HMI podría poner en riesgo aparatos con un coste mucho mayor, sin contar el tiempo que llevaría preparar y monitorizar los escenarios necesarios para validar esta herramienta.

Dada la importancia del usuario en el proceso de construcción de la herramienta y la complejidad en el diseño que esta presenta, se ha decidido incluir también en este proceso la evaluación de los requisitos de cada uno de los componentes de esta herramienta a través de la evaluación de la herramienta sobre el escenario propuesto por la competición IARC y distintas pruebas de usabilidad realizadas a los posibles futuros usuarios de la herramienta.

Finalmente, en un estado de desarrollo más avanzado de la aplicación se han realizado un conjunto de pruebas de validación que incluyen tanto pruebas unitarias a cada uno de los componentes que integra la interfaz como pruebas de integración que verifican la usabilidad, robustez, eficiencia y escalabilidad de la aplicación.

5.1 Evaluación de la herramienta sobre un dominio

Aunque esta herramienta ha sido diseñada con un propósito general, es necesaria la validación de la herramienta sobre un escenario real que verifique la aplicabilidad de la herramienta. Para esta evaluación se ha elegido el escenario propuesto por la competición IARC en la que actualmente participa el grupo de investigación de Sistemas Inteligentes e Ingeniería del Conocimiento junto con el grupo de investigación vision4UAV de la Universidad Politécnica de Madrid [24].

5.1.1 IARC – International Aerial Robotics Competition

IARC (*Internacional Aerial Robotic Competition*) [15] es la competición de robótica aérea más antigua y consolidada del mundo, fue creada para fomentar la investigación y el progreso en el campo de la robótica aérea. Su primera edición fue celebrada en 1991 en el Instituto Tecnológico de Georgia. Durante este tiempo, IARC ha contribuido

enormemente en la evolución de la tecnología robótica aérea, desde los inicios en los que encontrábamos robots aéreos que apenas podían mantenerse volando hasta los modernos vehículos aéreos prácticamente autónomos con la capacidad de interactuar con el entorno.

Debido a los logros obtenidos durante el tiempo, es considerada una de las competiciones con mayor relevancia en el mundo de la robótica. En ella han participado importantes universidades de todo mundo con el soporte económico de la industria y el gobierno. Las diferentes misiones se han diseñado contemplando un nivel creciente de complejidad, suponiendo un reto tecnológico en ese momento. Por ello es la propia competición quien se encarga de definir las reglas específicas y objetivos de cada misión, dejando hasta varias ediciones para llegar a completar alguna de las misiones. En total 6 de las misiones propuestas han sido completadas, encontrándose actualmente en la séptima y última misión.

A continuación se muestra la descripción general de la séptima misión.

5.1.2 Misión 7

El desarrollo de esta misión se realiza dentro de un cuadrilátero denominado "Arena" de 20 metros de lado. Este cuadrilátero está delimitado por líneas de 8 centímetros de grosor y de colores verde y rojo como se muestra en la Figura 5-1

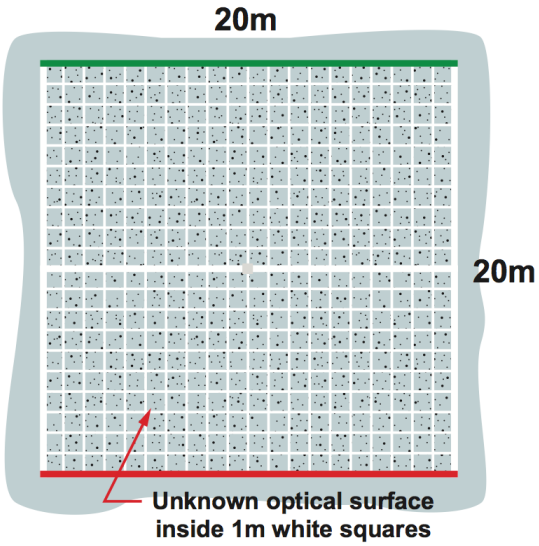


Figura 5-1. Arena de la séptima misión IARC.

Dentro de la Arena se dispondrán 10 robots terrestres autónomos de modelo "*iRobot Create*" que en el inicio se encontrar en el centro de la arena de tal manera que puedan moverse a cualquier dirección dentro del área delimitada.

El comportamiento de los robots de tierra es el siguiente. Una vez comienza la prueba empezaran a moverse hacia los límites de la arena, transcurridos 20 segundos o en caso de que ocurriese una colisión el robot cambiará su dirección 180º en el sentido de las agujas del reloj. Además, cada 5 segundos el robot variaría aleatoriamente su dirección en un rango de entre 0º y 20º. Cuando alguno de los robots cruza una de las líneas límites de la arena, será eliminado de la misión.

Cada uno de los robots de tierra posee en la parte superior un sensor magnético, cuando el UAV se acerque lo suficiente como para que el de tierra lo detecte, éste cambiará su dirección 45º en el sentido de las agujas del reloj. Se deberá tener en cuenta que dos descensos seguidos sobre el mismo robot genera un giro de 90º, es decir por cada descenso 45º de giro.

Además si el UAV desciende con el fin de cortar la trayectoria del robot de tierra, haciendo que éste colisione con él, el robot de tierra girará 180º en las agujas del reloj.

5.1.3 Pruebas realizadas

Para la validación del escenario propuesto por en la competición IARC en particular, se ha modificado ligeramente el simulador ya existente en el grupo de investigación para visualizar la misión IARC, realizando un cambio en la lectura de *topics* en el módulo de visualización de la arquitectura para visualizar el estado percibido a través de Rviz, este cambio se ha marcado con una flecha azul en la figura 5-2, introduciendo un error en los cambios de posición del UAV.

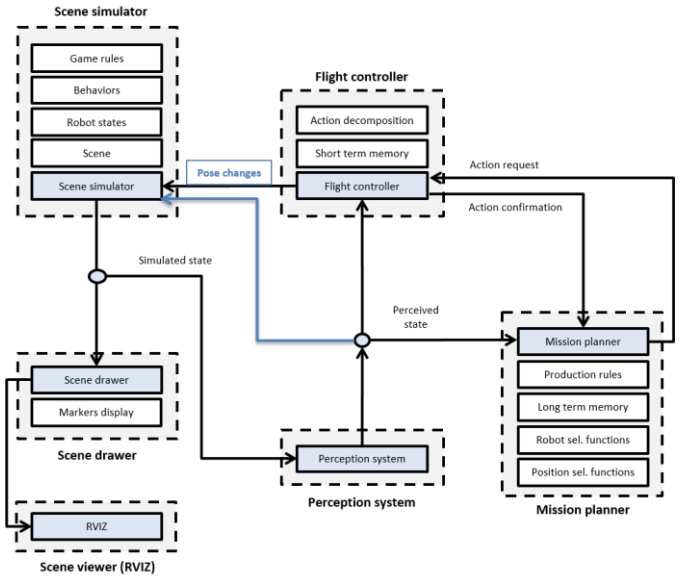


Figura 5-2. Arquitectura del simulador modificada

En el anexo B se puede ver un ejemplo del HMI funcionando con el simulador que implementa esta arquitectura. Para probar el correcto funcionamiento del HMI y la aplicabilidad de esta herramienta a este escenario, se han diseñado un conjunto de pruebas visuales destinadas a asegurar la correcta visualización de los principales requisitos propuestos de manera genérica en el capítulo 3 de este documento.

- **Comandos de control.** Para la realización de esta prueba se propusieron distintas trayectorias dentro del escenario de la misión IARC, asegurando que la ejecución y recepción de controles era similar al obtenido en el modo autónomo ejecutado por el simulador.

- **Visualización de la información de vuelo**. Para la realización de esta prueba se comprobó a través de la herramienta "rostopic echo" que la información mostrada principalmente en el visualizador de parámetros era correcta, informando al usuario de los parámetros recibidos en tiempo real.

- **Estado de la misión.** Las pruebas de este requisitos se han realizado actuando con el UAV a través de los dos modos principales de operación que proporciona el HMI, el modo manual y el modo autónomo, asegurando que en ambos casos se obtenía correctamente del sistema la acción actual que está ejecutando el UAV y que el visualizador del entorno percibido proporcionaba la abstracción del escenario que debería simular el sistema de acuerdo a los valores obtenidos en un sistema real.
- **Visualización de la dinámica del vehículo.** Estas pruebas se centraron en asegurar que la orientación y posición recibidas por el sistema coincida con la que se obtenía visualmente del simulador, operando con el vehículo en los modos manual y autónomo.

- **Estado del sistema.** Para probar este componente se han realizado pruebas mandando mensajes de error simulados para verificar que el sistema es capaz de reconocer los distintos tipos de error e informar correctamente al usuario a través del panel de control y el monitor de rendimiento del sistema.

6 Pruebas de validación software

Para la validación de la herramienta construida se ha utilizado el simulador desarrollado por el propio grupo de investigación de Sistemas Inteligentes e Ingeniería del Conocimiento de la Escuela Técnica Superior de Ingenieros Informáticos. Este simulador implementa una arquitectura simplificada de los subsistemas reales de la arquitectura original descrita en el capítulo 4 de este documento y objeto final de esta aplicación.

El uso de este simulador ha permitido validar los diferentes componentes de la interfaz de manera segura y controlada, llegando incluso a poder realizar pruebas unitarias cuando es necesario cambiando alguna de las salidas de estos procesos o variando significativamente alguno de los algoritmos o asunciones que se tuvieron en el proceso de construcción del mismo.

6.1.1 Pruebas unitarias

La realización de pruebas unitarias se ha realizado mediante la herramienta *rosconsole* que proporciona *ROS* para depuración de procesos. Además de estas herramientas previamente se han realizado pruebas con las herramientas gráficas de depuración que proporciona *ROS* para verificar que las entradas y salidas a los procesos que se estaban testeando no eran el origen del problema, llegando así a utilizar el resto de arquitectura como cajas negras que actúan sobre el proceso en el cual se están realizando las pruebas.

- **Comprobación de las entradas y salidas al proceso.** Está comprobación se ha realizado mediante la herramienta *rostopic*. Esta herramienta permite visualizar los mensajes enviados por un *topic* o canal de comunicación determinado mediante la opción "*echo"*. En la Figura 6-1 se muestra un ejemplo de este tipo de prueba.

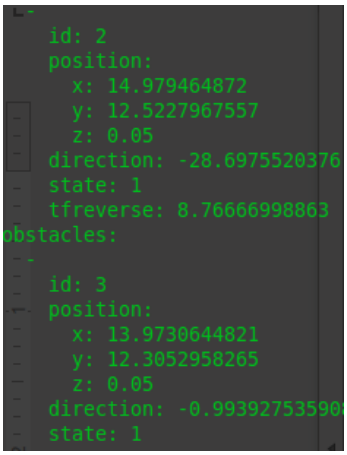


Figura 6-1. Rostopic echo

- **Comprobación del rango de frecuencias de los *topics***. Esta prueba ha sido necesaria sobre todo en el componente de visualización de parámetros para determinar la frecuencia de muestreo adecuada en la visualización de curvas gráficas. La realización de esta prueba se ha realizado también a través del comando *rostopic* con la opción "*hz*" como se puede observar en la Figura 6-2.

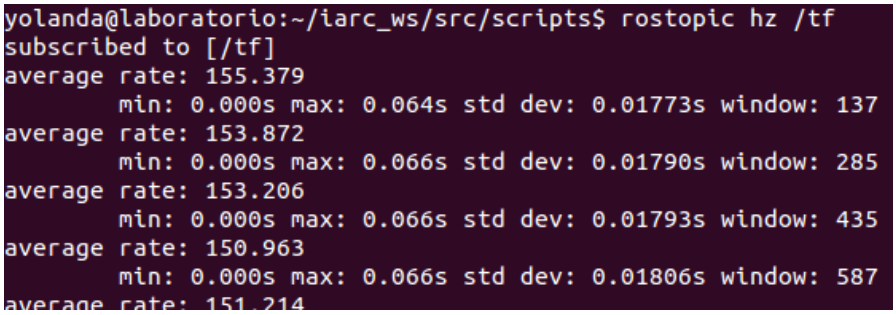


Figura 6-2. Rostopic hz /tf

- **Prueba de sincronización del proceso transmisor de comandos**. La realización de esta prueba se ha centrado en la coordinación de las señales emitidas por el usuario a través de la interfaz de comandos con la frecuencia de envío de comandos a través de ROS. Esta prueba es necesaria debido al carácter asíncrono que tienen la mayoría de los procesos en ROS, siendo indispensable en este caso asegurar una correspondencia de la frecuencia de publicación con la frecuencia media a la que se muestrean los eventos producidos por el teclado en la interfaz. Para la realización de esta prueba se ha utilizado la herramienta de depuración *rosconsole* a través de la macro instrucción `ROS_DEBUG_THROTTLE (period, ...)` imprimiendo el mensaje enviado asignando distintos valores de frecuencia.

- **Prueba de los procesos receptores de mensajes**. La realización de esta prueba se ha centrado en verificar que los mensajes son recibidos correctamente por cada uno de los procesos receptores de mensajes. Esta prueba se ha realizado mediante la macro instrucción ROS_INFO para la visualización de los mensajes, asignando a cada uno un identificador temporal a través de la librería de ROS con Ros::Time, verificando una coherencia temporal entre cada uno de los procesos (Figura 6-3). Finalmente se ha comprobado que el contenido de los mensajes era coherente para cada uno de los procesos con la macro instrucción `ROS_ASSERT_MSG (cond, ...)`.

```
[ INFO] [1435072227.628173763]: Received posy from Perceived State topic: [17,331472]
[ INFO] [1435072227.628201457]: Received posz from Perceived State topic: [0,514731]
[ INFO] [1435072227.628233899]: Received posx from Perceived State topic: [3,811561]
[ INFO] [1435072227.628247379]: Received posy from Perceived State topic: [17,472666]
[ INFO] [1435072227.628300489]: Received posz from Perceived State topic: [0,489474]
[ INFO] [1435072227.628430211]: Received posx from Perceived State topic: [4,469454]
[ INFO] [1435072227.628457298]: Received posy from Perceived State topic: [17,302683]
[ INFO] [1435072227.628468881]: Received posz from Perceived State topic: [0,200153]
```

Figura 6-3. ROS_INFO

- **Prueba de la tasa de refresco de las curvas gráficas**. La realización de esta prueba se ha realizado en el controlador de curvas gráficas, verificando una correspondencia entre la evolución temporal del eje X de la gráfica y una correcta lectura de valores y actualización de los mismos en las curvas de parámetros. Para la realización de estas pruebas se ha implementado un temporizador, mediante la librería *QTime* de la suite de *Qt*, al que se le han ido asignando distintos valores muestreo.

- **Pruebas realizadas a través de rosbag.** *Rosbag* es una herramienta que forma parte de la suite de ROS para grabar y reproducir la salida de los *topics* o canales de comunicación. Esta herramienta está pensada incluso para trabajar con distintas frecuencias evitando la deserialización y reserialización de mensajes con frecuencias superiores a la real a través de relojes simulados de ROS. Estas salidas se graban en un fichero denominado "bag" con un formato específico. El uso de esta herramienta ha permitido comprobar la salida de los componentes cercanos al hardware del UAV como podrían ser las cámaras o distintos sensores a bordo del UAV.

6.1.2 Pruebas de integración

Estas pruebas se han enfocado en dos conjuntos, el primero centrado en la usabilidad y el otro conjunto centrado en la integración con el sistema. Las pruebas de usabilidad han servido para medir la aceptación y capacidad de respuesta de la aplicación frente al usuario; mientras que las pruebas de integración con el sistema se han realizado para medir la escalabilidad, eficiencia y robustez de la aplicación a través de dos simuladores de una arquitectura autónoma con diferente nivel de complejidad y las diferentes herramientas que proporciona ROS.

Pruebas de usabilidad. Para la realización de estas pruebas se ha medido el tiempo y el número de *clicks* que le lleva al usuario encontrar y ejecutar las siguientes tareas básicas,

en la figura 6-3 se muestra un gráfico resultado de las pruebas, donde se puede observar que las tareas 1,2 y 5 han ocupado mayor porcentaje de tiempo/*clicks* para el usuario:

1. Conexión con el sistema.
2. Operación con el vehículo.
3. Visualización de parámetros.
4. Visualización de cámaras.
5. Cargar un fichero de configuración.
6. Visualización del estado de los procesos.
7. Visualización del entorno percibido.

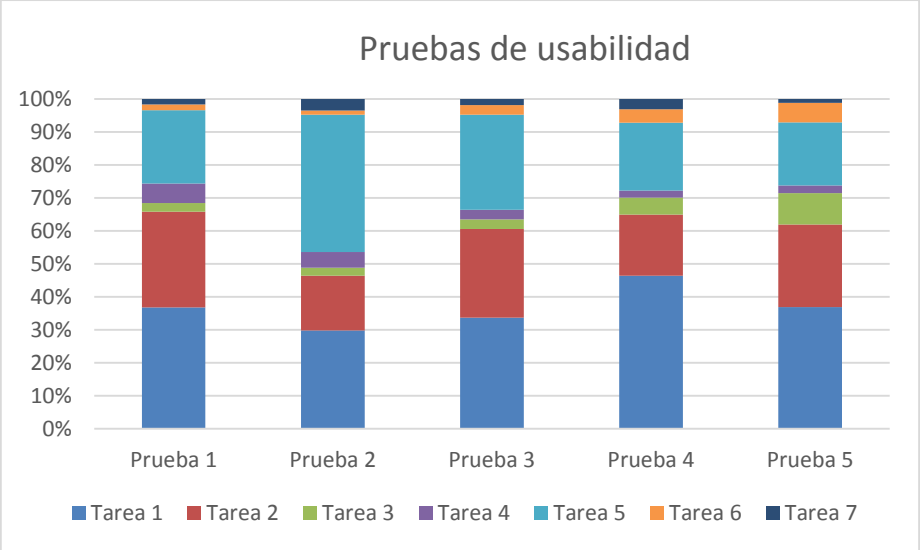


Figura 6-3. Pruebas de usabilidad

Pruebas de Integración. Para la realización de las pruebas de integración se han utilizado dos simuladores. El primero de ellos implementa una arquitectura autónoma simplificada. El segundo utiliza en la simulación algunos de los componentes de una arquitectura que actualmente ha sido probada en distintos escenarios de vuelo. Ambas arquitecturas están orientadas a cumplir los objetivos de la misión propuestos en la competición IARC (véase Anexo).

A continuación se presenta un grafo de los procesos obtenidos a través de la herramienta gráfica de ROS denominada *rqt_graph*, mostrando los nodos activos y conexiones con el HMI.

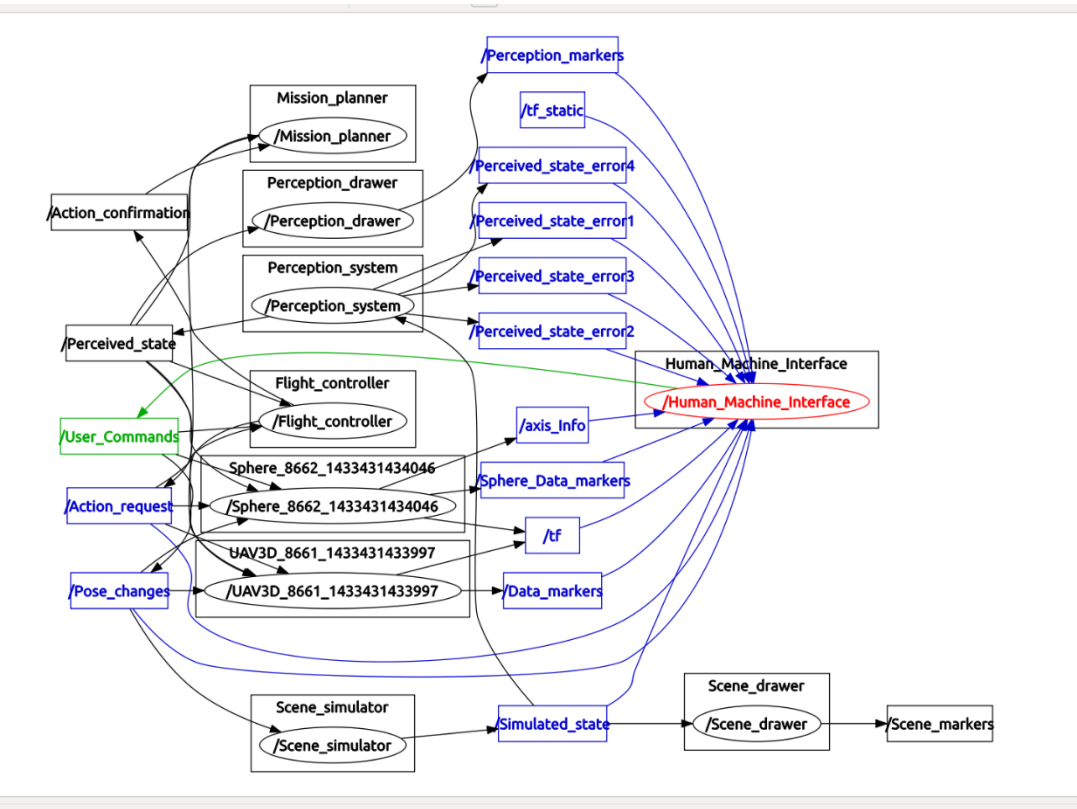


Figura 6-4. Rqt_graph

Además de esto se han comprobado que la salida de los mensajes fuera correcta y se recibieran en el mismo orden temporal, asignándolos y ordenándolos por el *TimeStamp* obtenido de la función Time de la API de ROS. Como se puede observar en la figura 6-5, el tipo de mensaje y las coordenadas x, y, z guardan siempre el mismo orden temporal.

human_machine_interface

```
[ INFO] [1435072223.123987192]: Received accelx from Pose Changes topic: [0,000000]
[ INFO] [1435072223.124001342]: Received accely from Pose Changes topic: [0,000000]
[ INFO] [1435072223.124013278]: Received accelz from Pose Changes topic: [0,000000]
[ INFO] [1435072223.141608227]: Received posx from Perceived State topic: [3,787234]
[ INFO] [1435072223.141627303]: Received posy from Perceived State topic: [17,568647]
[ INFO] [1435072223.141636797]: Received posz from Perceived State topic: [0,352104]
[ INFO] [1435072223.141971811]: Received posx from Perceived State topic: [3,948179]
[ INFO] [1435072223.141986600]: Received posy from Perceived State topic: [17,018478]
[ INFO] [1435072223.141995657]: Received posz from Perceived State topic: [0,500047]
```

Figura 6-5. Consola de comunicación

Además de comprobar la conexión entre los distintos componentes de la arquitectura se han realizado pruebas para verificar la independencia de la interfaz de usuario con el

sistema sobre el que opera para comunicarse con el software a bordo del UAV, esta independencia en el diseño software se puede comprobar en el diagrama de clases que figura en el Anexo de este documento. Para la realización de estas pruebas se ha lanzado la aplicación sin el nodo *roscore* (nodo central de ROS), obteniendo como salida de error "*El nodo roscore no ha sido inicializado*" mostrado en la aplicación a través de una ventana de aviso.

7 CONCLUSIONES Y TRABAJO FUTURO

Por último, en este capítulo se exponen las conclusiones que se han extraído de la realización de este Trabajo Fin de Grado, además de algunas de las propuestas de trabajos futuros a realizar.

Analizando los objetivos propuestos para la realización de este trabajo, descritos en el capítulo 1 de este documento, a continuación se detalla en qué medida se han satisfecho estos objetivos y las conclusiones extraídas de la realización del trabajo:

- La investigación en el campo de los UAV ha sido importante para determinar los factores influyentes en el diseño de la herramienta. Para realizar este estudio, se consideró importante revisar tanto las características técnicas de los UAV como el grado de interacción que estos pueden tener con el operador de la herramienta, atendiendo a los diferentes niveles de autonomía del vehículo y los diferentes modos de operación existentes. Todo ello permitió tener una idea general del problema y poder entender las posibles necesidades que podría tener el usuario de la herramienta.

- El estudio realizado durante la primera fase de análisis de las diversas soluciones de presentación para la operación con vehículos aéreos no tripulados, permitió partir de un diseño general que sirvió para luego especificar los requisitos de la herramienta. En el diseño de esta herramienta se buscó principalmente que fuera fácil de utilizar y que pudiera servir tanto al programador del sistema como a un usuario con un nivel conocimientos medio acerca de los UAV y de ROS en general. Además de este estudio, se realizó una evaluación de los factores humanos determinantes en el diseño de la herramienta y el modo en el que estos influían en el uso de la misma.

- En la implementación de la herramienta se eligió la librería gráfica *Qt* y el lenguaje de programación C++. Esta librería es utilizada ampliamente por la comunidad de programadores en la construcción de aplicaciones de escritorio, proporcionando una suite de librerías que permiten la programación de casi cualquier componente gráfico a través de un entorno de programación dirigido. Además de esta librería, se ha utilizado ROS para la comunicación con el sistema a bordo del UAV, permitiendo una abstracción de las características hardware y los sistemas de comunicación del UAV.

- En la fase de validación se han realizado un conjunto de pruebas, con el fin de que la herramienta cumpliera con los requisitos descritos en el capítulo 3 de este documento. Finalmente, tras las pruebas realizadas se consigue obtener una herramienta interacción persona-ordenador preparada para ser probada con un sistema real dentro de un entorno controlado.

En cuanto al desarrollo del cliente, se ha implementado una aplicación que permite visualizar todas las funcionalidades que ofrece el sistema autónomo a bordo del UAV y permite el uso de cualquier sistema autónomo que funcione sobre *ROS* en diferentes aplicaciones. Algunas de las aplicaciones que ofrece o podría llegar a ofrecer la aplicación son:

- Realizar un vuelo controlado a través de comandos enviados por la HMI. Ejemplos de estos comandos son: despegar (*take off*), ir hacia delante (*forward*), ir hacia detrás (*backward*), aterrizar (*land*), etc. El operador utiliza el teclado y el ratón para controlar el UAV a través de la HMI.

- Seguir un objeto seleccionado a través de contacto visual. El usuario puede seleccionar el objeto a seguir a través de las imágenes recibidas por la cámara o mediante la definición de objetivos de alto nivel a través de la HMI.

- Planificación de la misión, a través de una serie de *waypoints* proporcionados a través de un fichero XML configurable a través de la HMI.

- Visualización a través de la HMI de una recreación 3D de zonas inaccesibles por el humano a través del conjunto de sensores de UAV y de los algoritmos de percepción.

- Recepción de imágenes y/o vídeo de zonas inaccesibles por el humano a través del conjunto de cámaras a bordo del vehículo.

- Volar en un área específica, estableciendo unos límites con un fichero XML configurable a través de la HMI.

LISTA DE ACRÓNIMOS

HMI	Human Machine Interface
UPM	Universidad Politécnica de Madrid
UAV	Unmanned Aerial Vehicle
STANAG	NATO Standardization Agreement
UAS	Unmanned Aircraft Systems
JCGUAS	NATO Joint Capability Group On Unmanned Aerial System
GPS	Global Position Satellite
IMINT	Imagery intelligence
COMINT	Communications Intelligence
ELINT	Electronic signals intelligence
SIGINT	Signals Intelligence
ACL	Autonomous Control Level
AFRL	Air Force Research Laboratory
ALFUS	Autonomy Levels For Unmanned Systems
ALFURS	Autonomy Levels For Unmanned Rotorcraft Systems
EC	Environmental Complexity
AEF	Autonomy Enabling Functions
HI	Human Independence
IARC	International Aerial Robotics Competition
MC	Mission Complexity
NIST	National Institute of Standards and Technology
MIT	Massachusetts Institute of Technology
OODA	Observe, Orient, Decide and Act
PR	Personal Robots
RAM	Random-Access Memory
ROS	Robot Operating System
RUAS	Rotorcraft Unmanned Aerial System
GUI	Graphical User Interface
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
URDF	Universal Robotic Description Format
XML	eXtensible Markup Language

8 BIBLIOGRAFIA

[1] Unmanned aerial vehicles. [En línea] Disponible en: http://en.wikipedia.org/wiki/Unmanned_aerial_vehicle. [Último acceso: Mayo 2015]

[2] STANAG 4586 (Edition 2) Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability – 2007.

[3] G. Crespo, "Sistema de enlace robusto para la teleoperación de UAV (vehículo aéreo no tripulado)," Trabajo Fin de Grado UAM, 2014

[4] A. M. Alcaraz, "Sistema de soporte al desarrollo de un planificador de misiones en un vehículo aéreo no tripulado," Trabajo Fin de Grado UPM, 2014.

[5] L.Damilano, "An Innovative Human Machine Interface for UAS Flight Management System", 2012.

[6] T. B. Sherindan, Telerobotics, Automation, and Human Supervisory, MIT Press, 1992.

[7] T. B. S. F. I. a. C. D. W. Raja Parasuraman, «A Model for Types and Levels of Human Interaction»

[8] F. Kendoul, «A Survey of Advances in Guidance, Navigation and Control of Unmanned Rotorcraft Systems,» 2012.

[9] QGroundControl [En línea] http://qgroundcontrol.org/start [Último acceso: Mayo 2015]

[10] Sanchez-Lopez, J.L.,; Pestana, J.; de la Puente, P.; Campoy, P. (2015): "Reusable Open-source System Architecture for the Fast Designing and Prototyping of Autonomous Multi-UAV Systems: Simulation and Experimentation". Journal of Intelligent and Robotic Systems.

[11] R. Murphy, Introduction to AI Robotics, 2000.

[12] Arkin, R. C., Riseman, E. M., and Hansen, A. (1987): "AuRA: An Architecture for Vision- Based Robot Navigation," proceedings of the DARPA Image Understanding Work-shop, Los Angeles, CA, February, pp. 417–413.

[13] Gat , E. (1998): "On Three-Layer Architectures". In "Artificial Intelligence and Mobile Robots", (David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, eds.) AAAI Press.

[14] IARC, «Official Rules for the International Aerial Robotics Competition Mission 7 v11.0,» 2014.

[15] M. R. Endsley, «Situation awareness in aviation systems,» *Handbook of Aviation Human Factors, Publication of Lawrence Erlbaum Associates*, 1999

[16] Jill L. Drury, Laurel Riek, Nathan Rackliffe .A Decomposition of UAV-Related Situation Awareness.

[17] M.R. Endsley. Toward a Theory of situation awareness in Dynamic Systems.Texas Tech University, Lubbock.

[18] M. R. Endsley, Design and Evaluation for Situation Awareness Enhancement, 1988.

[19] Open Source Robotics Foundation, «ROS - Robot Operating System,» [En línea]. Available: http://www.ros.org/. [Último acceso: February 2015].

[20] Biblioteca Qt. [En línea] http://es.wikipedia.org/wiki/Qt_%28biblioteca%29 [Último acceso: Junio 2015]

[21] T. Foote, «tf: The Transform Library,» 2013.

[22] A Survey of Advances in Guidance, Navigation and Control of Unmanned Rotorcraft Systems.

[23] M.A.Catalán Vega.Metodología de evaluación de Interfaces Gráficas de Usuario.

[24] J Pestana, JL Sanchez-Lopez, R Suarez-Fernandez, JF Collumeau, P Campoy, J Martin-Cristobal, M Molina, J De Lope, and D Maravall. A vision based aerial robot solution for the IARC 2014 by the Technical University of Madrid. In 2014 International Aerial Robotics Competition, 2014.

ANEXO A: Prototipos exploratorios

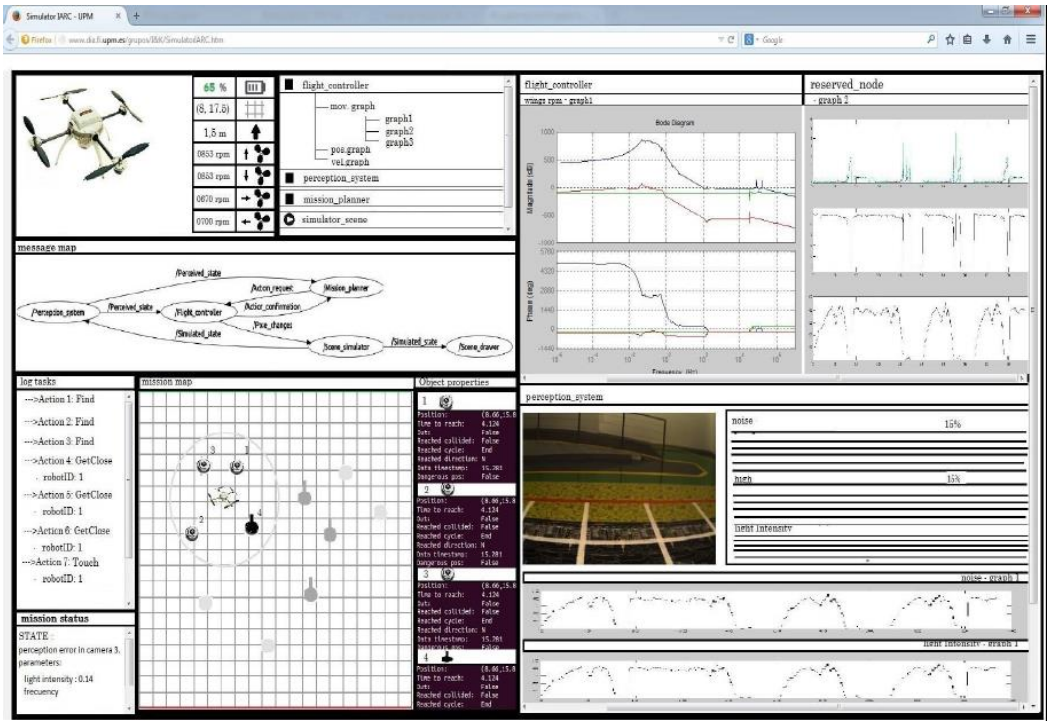


Figura A-1. Prototipo 1

Figura A-2. Prototipo 2

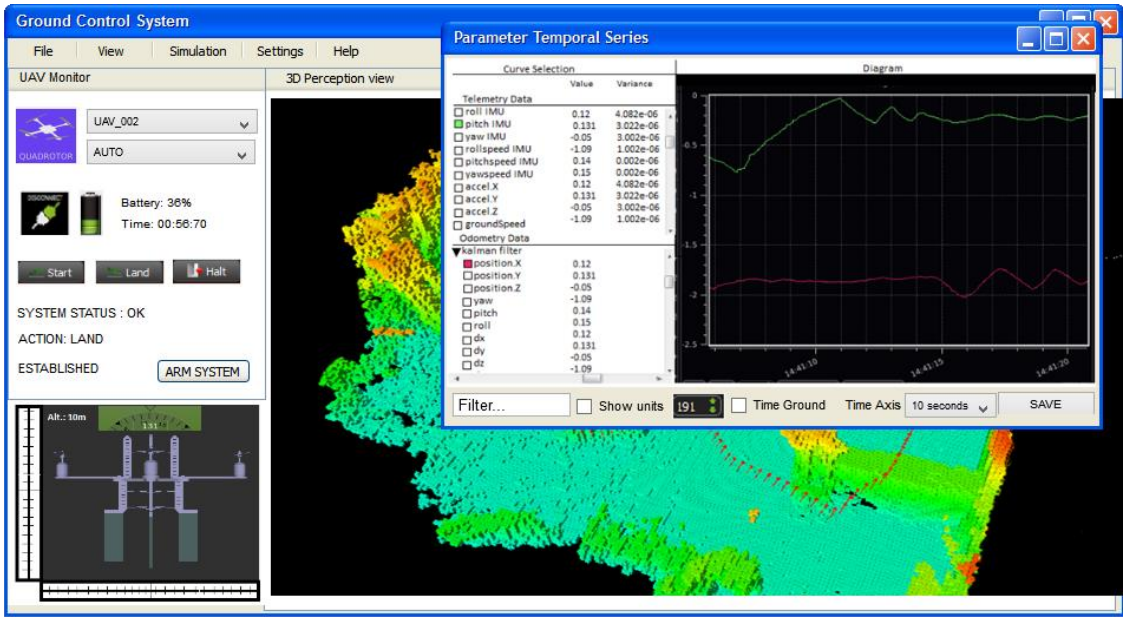


Figura A-3. Prototipo final.

ANEXO B: Pruebas realizadas con simuladores

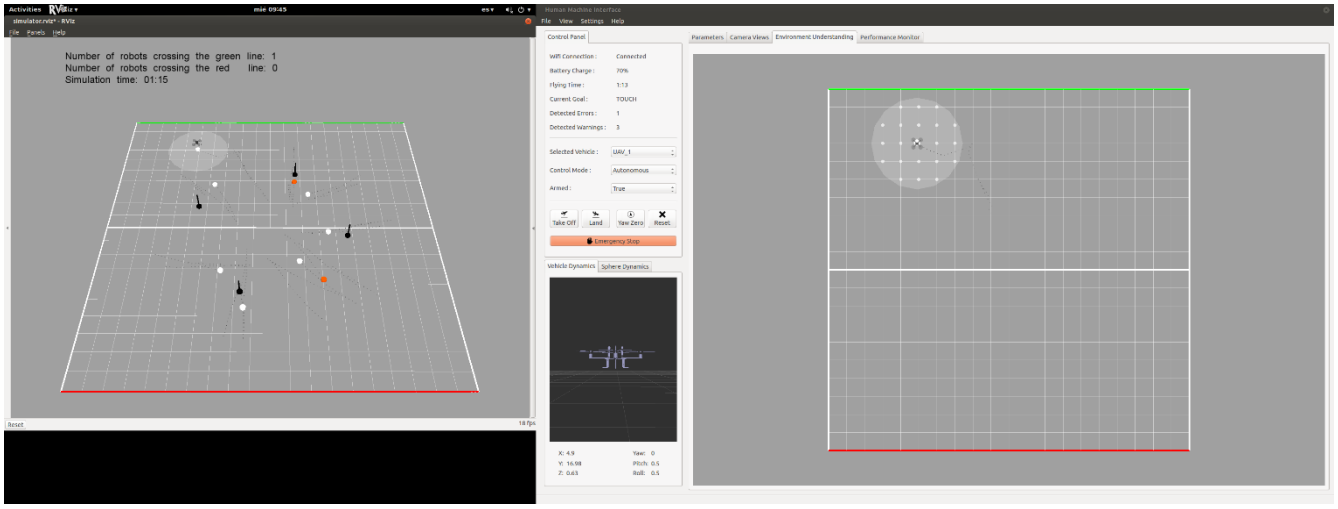


Figura B-1. Pruebas de integración con el Simulador 1.

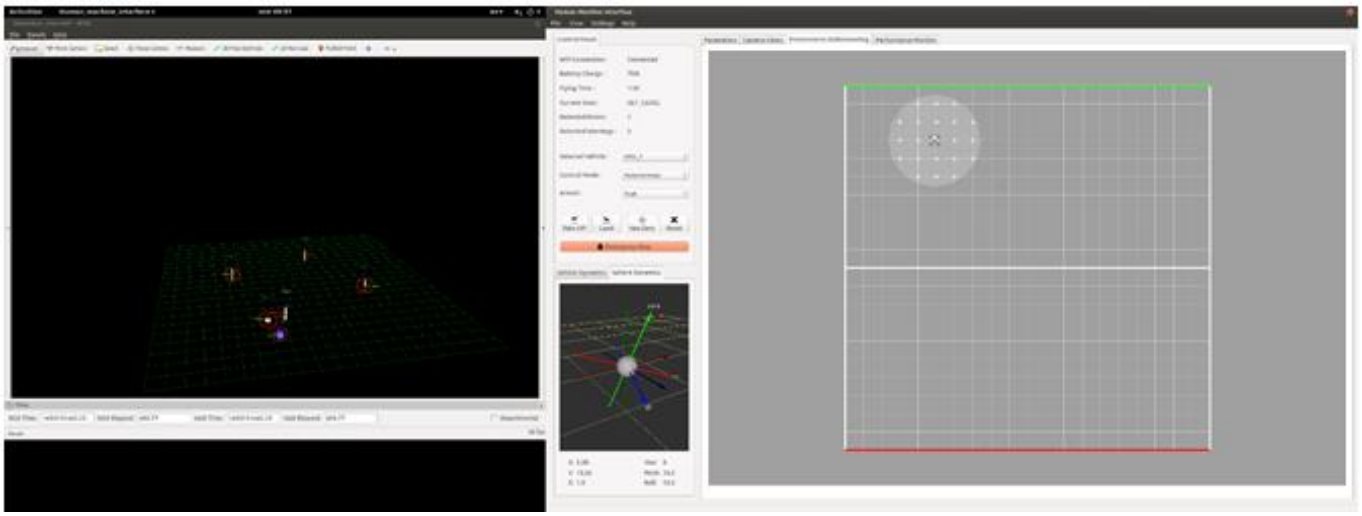


Figura B-2. Pruebas de integración con el Simulador 2.

ANEXO C: Diagrama de clases

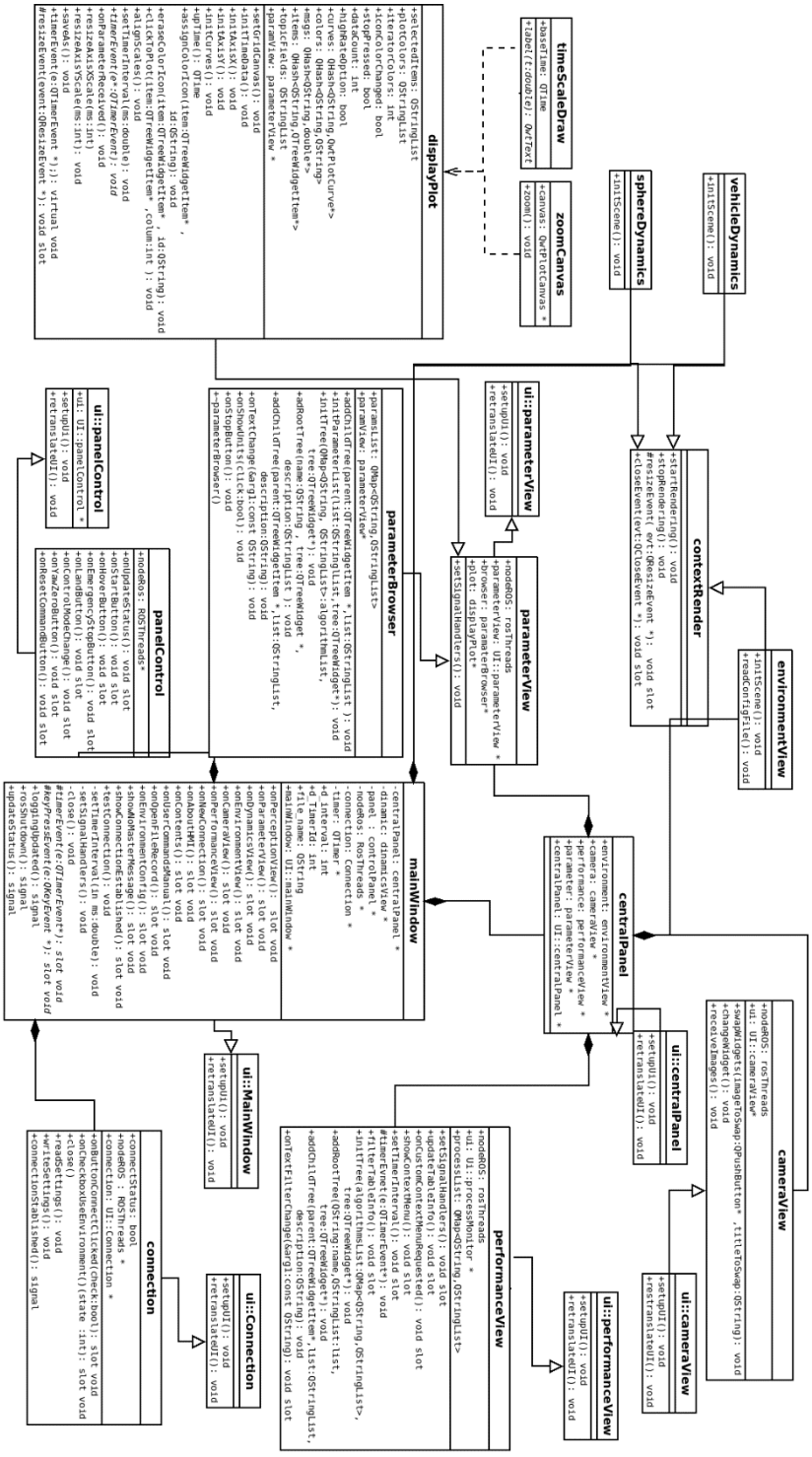


Figura C-1 Diseño software de la capa de interfaz de usuario

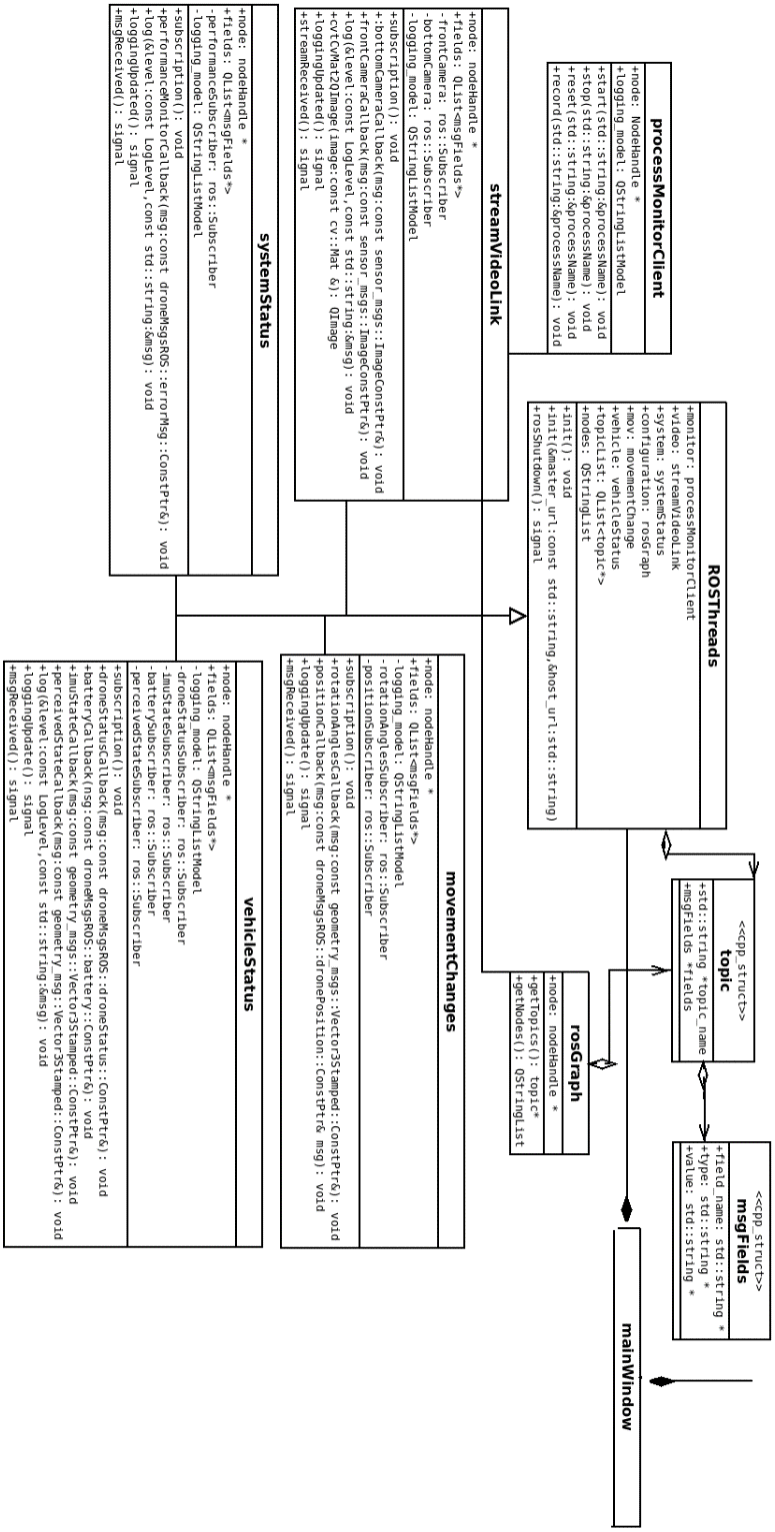


Figura C-2. Diagrama de clases de la capa de comunicación

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Thu Jun 25 23:25:31 CEST 2015
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)